CENTRO PAULA SOUZA

ETEC EURO ALBINO DE SOUZA

Ensino Técnico Integrado ao Médio de Desenvolvimento de Sistemas

Ana Laura Martins de Almeida

Camila de Moraes Rocha

Guilherme Pereira

Joyce Cristina da Silva

Juan Andres Sanchez de Sousa

Vinícius Alberto Ramalho

BIBLIONTEC

Mogi Guaçu – SP

2025

Ana Laura Martins de Almeida

Camila de Moraes Rocha

Guilherme Pereira

Joyce Cristina da Silva

Juan Andres Sanchez de Sousa

Vinícius Alberto Ramalho

BIBLIONTEC

Trabalho de Conclusão de Curso apresentado ao Curso Ensino Técnico Integrado ao Médio em Desenvolvimento de Sistemas da ETEC Euro Albino de Souza, orientado pelo Prof. Maércio Girardi Bisco e Pedro Ramires da Silva Amalfi Costa, como requisito parcial para a obtenção do título de técnico em Desenvolvimento de Sistemas.

Mogi Guaçu – SP

2025

A447b

Almeida, Ana Laura Martins de; Rocha, Camila de Moraes; Pereira, Guilherme; Silva, Joyce Cristina da; Sousa, Juan Andres Sanchez de; Ramalho, Vinícius Alberto.

BibliONtec. – Mogi Guaçu – SP. 2025.

289 p.

Trabalho de Conclusão do Curso de Ensino Médio com Habilitação Profissional de Técnico em Desenvolvimento de Sistemas da ETEC Euro Albino de Souza de Mogi Guaçu.

Orientadores: Professor Especialista Maércio Girardi Bisco e
Professor Mestre Pedro Ramires da Silva Amalfi Costa

1. Biblioteca. 2. Organização. 3. Modernização. I. Bisco, Maércio Girardi; Costa, Pedro Ramires da Silva Amalfi (orientadores) II. Título.

CDD 027.8

Bibliotecário Guerreiro Ímpar Nunes Guerreiro – CRB-8/8137

DEDICATÓRIA

Dedico este trabalho com imensa gratidão ao professor Reginaldo Donizete Candido, que, mesmo sem lecionar para nós, generosamente dedicou seu tempo para nos apoiar. E ao amigo José Gabriel Cezário Barreto, cuja ajuda constante e parceria fizeram toda a diferença. A ambos, meu sincero muito obrigado!

AGRADECIMENTO

Aos nossos pais, agradecemos profundamente por serem nossa base, inspirando-nos e nos apoiando com amor incondicional em cada passo desta jornada. Vocês sempre estiveram ao nosso lado, oferecendo força nos momentos difíceis e comemorando cada conquista conosco. Aos amigos, nosso sincero obrigado pela amizade, parceria e incentivo constante. Cada palavra de apoio e cada gesto de carinho fizeram toda a diferença. A todos, nossa eterna gratidão.

EPÍGRAFE

"É preciso que a leitura seja um ato de amor." Paulo Freire

RESUMO

BibliONtec é um projeto de TCC (trabalho de conclusão de curso) que busca desenvolver um sistema *online* para modernizar a administração da biblioteca escolar. A ideia é aperfeiçoar o gerenciamento do acervo e das tarefas administrativas, tornando tudo mais rápido, organizado e acessível para bibliotecários e estudantes. A plataforma terá um catálogo virtual de livros, disponível para consulta dos usuários, e uma área administrativa exclusiva para os responsáveis pela biblioteca, onde poderão registrar novos livros, controlar os empréstimos e monitorar os prazos de entrega. O sistema será projetado com um visual amigável e adaptável, assegurando uma navegação simples e eficiente em diversos aparelhos. A motivação veio da análise dos desafios na organização e no funcionamento da biblioteca, evidenciando a importância de um sistema atual, seguro e eficiente que auxilie diretamente na gestão da escola.

Palavras-chave: Biblioteca; Organização; Modernização; Sistema *Online*; Acessibilidade.

ABSTRACT

BibliONtec *is a final project aimed at developing an online system to modernize the management of the school library. The goal is to improve the organization of the collection and administrative tasks, making everything faster, more organized, and accessible for librarians and students. The platform will feature a virtual book catalog available for user consultation and an exclusive administrative area for library staff, where they can register new books, manage loans, and monitor return deadlines. The system will be designed with a user-friendly and responsive interface, ensuring simple and efficient navigation across different devices. The motivation came from analyzing the challenges in organizing and operating the library, highlighting the importance of a modern, secure, and efficient system that directly supports school management.*

Key words**:** *Library; Organization; Modernization; Online System; Accessibility.*

LISTA DE ILUSTRAÇÕES

Figura 1 - Pergunta 1 Questionário 65
Figura 2 - Pergunta 2 Questionário 66
Figura 3 - Pergunta 3 Questionário 67
Figura 4 - Pergunta 4 Questionário 67
Figura 5 - Pergunta 5 Questionário 68
Figura 6 - Pergunta 6 Questionário 69
Figura 7 - Pergunta 7 Questionário 69
Figura 8 - Pergunta 8 Questionário 70
Figura 9 - Tela *Login* Protótipo 93
Figura 10 - Tela Redefinir senha Protótipo 94
Figura 11 - Tela Confirmar redefinir senha Protótipo 94
Figura 12 - Tela Inicial Protótipo 95
Figura 13 - Tela Indicações Protótipo 96
Figura 14 - Tela Lista de desejos Protótipo 97
Figura 15 - Tela Área do aluno Protótipo 98
Figura 16 - Tela Meus livros Protótipo 99
Figura 17 - Tela Eventos Protótipo 99
Figura 18 - Tela Detalhes do livro Protótipo 100
Figura 19 - Tela de reserva Protótipo 101
Figura 20 - Tela Adicionar lista desejos livro Protótipo 102
Figura 21 - Tela Lista de livros Protótipo 103
Figura 22 - Tela inicial bibliotecário 104
Figura 23 - Tela Trocar conta 105
Figura 24 - Tela Atualizações 106
Figura 25 - Tela Cadastro de alunos protótipo 106
Figura 26 - Tela Cadastro de livros Protótipo 107
Figura 27 - Tela Cadastro de funcionários Protótipo 108
Figura 28 - Tela Empréstimo Protótipo 109
Figura 29 - Tela Etiqueta Protótipo 109
Figura 30 - Tela Área Administrador Protótipo 110
Figura 31 - Tela Acervo Protótipo 111
Figura 32 - Tela Editar livro Protótipo 112
Figura 33 - Tela Usuários cadastrados Protótipo 113
Figura 34 - Tela Editar Usuário Protótipo 114
Figura 35 - Tela Funcionários cadastrados Protótipo 115
Figura 36 - Tela Editar funcionários Protótipo 116
Figura 37 - Tela *Backup* Protótipo 117
Figura 38 - BPMN do Administrador 145
Figura 39 - BPMN do Aluno 146
Figura 40 - BPMN do Bibliotecário 148
Figura 41 - BPMN do Professor 149
Figura 42 - Diagrama do Aluno 151
Figura 43 - Diagrama do administrador 152

Figura 44 - Diagrama do Bibliotecário 154
Figura 45 - Diagrama do Professor 155
Figura 46 - Diagrama de Classe 157
Figura 47 - Tela de *login* 159
Figura 48 - Código de *Login* 160
Figura 49 – Conexão com o banco de dados *login* 162
Figura 50 - Tela Inicial bibliotecário 164
Figura 51 - Código HTML da Tela Inicial 166
Figura 52 - Menu lateral do sistema 167
Figura 53 - Menu lateral expandido 169
Figura 54 - Código do menu lateral 171
Figura 55 - Tela área bibliotecário 172
Figura 56 - Código para carregamento de usuário A 174
Figura 57 - Código para carregamento de usuário B 175
Figura 58 - Conexão com banco de dados carregamento de usuários 176
Figura 59 - Tela de cadastro de usuários 178
Figura 60 - Código de cadastro de usuários A 179
Figura 61 - Código de cadastro de usuários B 180
Figura 62 - Conexão com banco de dados cadastro de usuário 182
Figura 63 - Tela cadastro de livros 183
Figura 64 - Código cadastro de livros A 185
Figura 65 - Código de cadastro de livros B 186
Figura 66 - Conexão com banco de dados cadastro de livros A 188
Figura 67 - Conexão com banco de dados cadastro de livros B 189
Figura 68 - Tela de cadastro de funcionários 191
Figura 69 - Código cadastro de funcionários 192
Figura 70 - Conexão com banco de dados cadastro funcionário 193
Figura 71 - Tela de empréstimo pesquisar usuário 194
Figura 72 - Tela empréstimo pesquisar livro 195
Figura 73 - Tela de empréstimo indisponível 196
Figura 74 - Código de empréstimo A 197
Figura 75 - Código de empréstimo B 198
Figura 76 - Código de empréstimo C 199
Figura 77 - Conexão com banco de dados empréstimo A 201
Figura 78 - Conexão com banco de dados empréstimo B 202
Figura 79 - Conexão com banco de dados empréstimo C 203
Figura 80 - Tela acervo de livros 205
Figura 81 - Tela *modal* editar livro 206
Figura 82 - Tela *modal* etiqueta 207
Figura 83 - Código acervo de livros A 208
Figura 84 - Código acervo de livros B 209
Figura 85 - Código acervo de livros C 210
Figura 86 - Conexão com o banco de dados acervo de livros A 213
Figura 87 - Conexão com o banco de dados acervo de livros B 214
Figura 88 - Tela usuários cadastrados 216
Figura 89 - Tela *modal* usuários cadastrados 217
Figura 90 - Código usuários cadastrados 218
Figura 91 - Conexão com o banco de dados atualizar dados 220

Figura 92 - Tela funcionários cadastrados 221
Figura 93 - Código funcionários cadastrados A 222
Figura 94 - Código funcionários cadastrados B 223
Figura 95 - Conexão com o banco de dados funcionários cadastrados 225
Figura 96 - Tela Inicial Administrador 226
Figura 97 - Tela área administrador 228
Figura 98 - Tela de Administração dos funcionários 229
Figura 99 - Tela *modal* de administração editar funcionários 230
Figura 100 - Código Administração de funcionários A 231
Figura 101 - Código Administração de funcionários B 232
Figura 102 - Tela configuração de prazos e regras 234
Figura 103 - Código configuração de prazos e regras 235
Figura 104 - Conexão com banco de dados configuração de prazos e regras 236
Figura 105 - Tela configuração de permissões 237
Figura 106 - Código configuração de permissões 239
Figura 107 - Conexão com banco de dados configuração de permissões 240
Figura 108 - Tela configurações da instituição 241
Figura 109 - Código configurações da instituição 242
Figura 110 - Conexão com banco de dados configurações da instituição 243
Figura 111 - Tela configurações de notificação 245
Figura 112 - Código configurações de notificação 246
Figura 113 - Conexão com banco de dados configurações de notificação 247
Figura 114 - Tela inicial do professor 248
Figura 115 - Tela do professor área dos livros 249
Figura 116 - Tela área do professor 250
Figura 117 - Conexão com o banco de dados da tela do professor 251
Figura 118 - Tela do professor indicar livro 253
Figura 119 - Código de indicação do *modal* 254
Figura 120 - Conexão com banco de dados do indicador 256
Figura 121 - Tela de *modal* para indicar livros 257
Figura 122 - Código do *modal* indicar livro 259
Figura 123 - Conexão com banco de dados do *modal* 261
Figura 124 - Tela de desindicar livro 263
Figura 125 - Conexão com banco de dados do desindicar livro 265
Figura 126 - Tela Inicial Aluno 267
Figura 127 - Tela Área do Aluno 268
Figura 128 - Tela livros disponíveis 269
Figura 129 - Tela comentários 271
Figura 130 - Código comentários 273
Figura 131 - Conexão com banco de dados comentários 275

LISTA DE TABELAS

Tabela 1 - Cronograma 52
Tabela 2 - Viabilidade Econômica 54
Tabela 3 - Custos 124

LISTA DE ABREVIATURAS E SIGLAS

ALGOL *(ALGOrithmic Language)* – Linguagem Algorítmica

API (*Application Programming Interfaces*) – Interface de Programação de Aplicações

APP *(Application)* – Aplicativo

ARPANET *(Advanced Research Projects Agency Network)* – Rede da Agência de Projetos de Pesquisa Avançada

AWS - Amazon Web Services

BDOO – Banco de Dados Orientado a Objetos

BPMN (*Business Process Model and Notation*) - Modelo e Notação de Processos de Negócio

CD (*Compact Disc*) - Disco Compacto

CDC - Código de Defesa do Consumidor

CDD - Classificação Decimal de Dewey

CERN *(Conseil Européen pour la Recherche Nucléaire)* – Organização Europeia para Pesquisa Nuclear

CGI *(Common Gateway Interface)* – Interface de *Gateway* Comum

CLT- Consolidação das leis trabalhistas

CNPJ – Cadastro Nacional de Pessoa Jurídica

COBOL *(COmmon Business-Oriented Language)* – Linguagem Orientada a Negócios Comum

CORBA (*Common Object Request Broker Architecture*) - Arquitetura Comum de Corretor de Requisições de Objetos

CPF – Cadastro de Pessoa Física

CSS *(Cascading Style Sheets)* – Folhas de Estilo em Cascata

DB *(DataBase)* – Banco de Dados

DBMS *(Database Management System)* – Sistema de Gerenciamento de Banco de Dados

DMN (*Decision Model and Notation*) – Modelo e Notação de Decisão

DNS *(Domain Name System)* – Sistema de Nomes de Domínio

DOM (*Document Object Model*) – Modelo de documento por objetos

ECA- Estatuto da criança e do adolescente

EER *(Extended Entity-Relationship)* – Entidade-Relacionamento Estendida

ER – Entidade-Relacionamento

ES – ECMAScript

FORTRAN *(FORmula TRANslation)* – Tradução de Fórmulas

FTP *(File Transfer Protocol)* – Protocolo de Transferência de Arquivos

GPD – Grupo de Pesquisa em Banco de Dados

HQ - História em Quadrinhos

HTML *(Hypertext Markup Language)* – Linguagem de Marcação de Hipertexto

HTTP *(Hypertext Transfer Protocol)* – Protocolo de Transferência de Hipertexto

HTTPS *(Hypertext Transfer Protocol Secure)* – Protocolo de Transferência de Hipertexto Seguro

IBM *(International Business Machines)* – Máquinas de Negócios Internacionais

ID – Código Identificador

iDE *(Integrated Development Environment)* – Ambiente de Desenvolvimento Integrado

IoT *(Internet of Things)* – *Internet* das Coisas

ISBN (*International Standard Book Number*) - Número Padrão Internacional para Livros

JS – JavaScript

KHTML *(KDE Hypertext Markup Language)* – Linguagem de Marcação de Hipertexto da KDE

KPI (*Key Performance Indicators*) – Indicadores-Chave de Desempenho

LAI- Lei de acesso à informação

LAN *(Local Area Network)* – Rede de Área Local

LDB - Lei de diretrizes e bases de educação

LGPD- Lei geral de proteção de dados

LISP *(LISt Processing)* – Processamento de Listas

MAC – Macintosh

MARC (*Machine Readable Cataloging*) - Catalogação Legível por Computador

MT – Mato Grosso

NAS *(Network Attached Storage)* – Armazenamento Conectado em Rede

NCSA *(National Center for Supercomputing Applications)* – Centro Nacional para Aplicações de Supercomputação

NoSQL *(Not Only SQL)* – Não Apenas SQL

NPM (*Node Package Manager*) – Gerenciador de Pacotes Nodes

NVM (*Node Version Manager*) – Gerenciador de Versões para Node

OCEB (OMG *Certified Expert in BPM)* - Especialista Certificado em BPM pelo OMG

OCRES (OMG *Certified Real-time and Embedded Specialist*) - Especialista Certificado em Sistemas de Tempo Real e Embarcados pelo OMG

OCSMP (OMG *Certified Systems Modeling Professional*) - Profissional Certificado em Modelagem de Sistemas pelo OMG

OCUP (OMG *Certified* UML *Professional*) - Profissional Certificado em UML pelo OMG

OMA (*Object Management Architecture*) - Arquitetura de Gerenciamento de Objetos

OMG (*Object Management Group*) - Grupo de Gerenciamento de Objetos

ORB (*Object Request Broker*) - Corretor de Requisições de Objetos

PHP *(Hypertext Preprocessor)* – Pré-processador de Hipertexto

Prolog *(Programming in Logic)* – Programação em Lógica

RAM *(Random Access Memory)* – Memória de Acesso Aleatório

RDBMS *(Relational Database Management System)* – Sistema de Gerenciamento de Banco de Dados Relacional

REST (*Representational State Transfer*) – Transferência de Estado Representacional

RFC *(Request for Comments)* – Solicitação de Comentários

RM - Registro de Matrícula

RMBE-Rede Municipal de Bibliotecas Escolares

RS – Rio Grande do Sul

SAP *(Systems, Applications, and Products in Data Processing)* – Sistemas, Aplicações e Produtos em Processamento de Dados

SGBD – *S*istema de Gerenciamento de Banco de Dados

SMS *(Short Message Service)* – Serviço de Mensagens Curtas

SQL *(Structured Query Language)* – Linguagem de Consulta Estruturada

SQL/DS *(Structured Query Language/Data System)* – Linguagem de Consulta Estruturada/Sistema de Dados

SSD *(Solid State Drive)* – Unidade de Estado Sólido

SSL - (*Secure Sockets Layer* – Camada de Soquete Seguro)

SysML (*Systems Modeling Language*) - Linguagem de Modelagem de Sistemas

TB – *Terabyte*

TCC - Trabalho de conclusão de curso

TELNET *(Telecommunication Network)* – Rede de Telecomunicações

TI - Tecnologia da Informação

TLS - (*Transport Layer Security* – Segurança de Camada de Transporte)

UFRN – Universidade Federal do Rio Grande do Norte

UFSC – Universidade Federal de Santa Catarina

UI *(User Interface)* – Interface do Usuário

UML (*Unified Modeling Language*) - Linguagem de Modelagem Unificada

UNIVAG – Universidade do Vale do Itajaí

URL *(Uniform Resource Locator)* – Localizador Uniforme de Recursos

UX (*User Experience*) - Experiência do Usuário

VIP (*Very Important Person*) – Pessoa Muito Importante

VS – Visual Studio

W3C *(World Wide Web Consortium)* – Consórcio da *World Wide Web*

WAN *(Wide Area Network)* – Rede de Área Ampla

WAP *(Wireless Application Protocol)* – Protocolo de Aplicação Sem Fio

WCAG - Diretrizes de acessibilidade para o conteúdo na *web*

WWW *(World Wide Web)* – Rede Mundial de Documentos Interligados

XAMPP – X (multiplataforma), Apache, MySQL, PHP, Perl

SUMARIO

INTRODUÇÃO 20
1 – TECNOLOGIAS 21
1.1 - LÓGICA DE PROGRAMAÇÃO 21
1.2 - Linguagem de Programação 21
1.2.1 – *Desktop* 23
1.2.2 – *Mobile* 23
1.2.3 – *Web* 25
1.3 – Servidores 26
1.3.1 – Locais 27
1.3.2 – *Web* 28
1.4 – Navegadores 28
1.5 - *Sites* Estáticos 30
1.6 – HTML 31
1.7 – CSS 31
1.8 – *SITES* DINÂMICOS 32
1.9 – JAVASCRIPT 32
1.10 – PHP 33
1.11 – Node.js 34
1.12 – Visual Studio code 35
1.13 - GITHUB 37
1.14 – Banco de Dados 38
1.14.1 – Modelagem 39
1.14.2 – Modelo conceitual 41
1.14.3 – Modelo lógico 42
1.14.4 – Modelo físico 42
1.15 – BR MODELO 42
1.16 – XAMPP 43
1.17 – MYSQL 44
1.17.1 – MariaDB 44
1.18 – CANVA 45
1.19 – FIGMA 47
2 – ESTUDO DE CASO 48
2.1 – Viabilidades 49

2.1.1 – Viabilidade operacional 49
2.1.2 - Viabilidade técnica 50
2.1.3 - Viabilidade de cronograma 52
2.1.4 - Viabilidade de econômica 54
2.2 – Técnica de levantamento de requisitos 55
2.2.1 – Entrevista 55
2.2.2 – Questionário 63
2.2.3 - Análise de texto 65
2.3 – Prototipagem 93
2.3.1 – *Site Web* – Aluno 93
2.3.2 – *Site Web* – Bibliotecário 104
2.4 – Clientes 117
2.4.1 – Cliente Interno 117
2.4.2 – Cliente Externo 118
2.4.3 – Cliente Ativo 120
2.4.4 – Cliente Inativo 121
2.4.5 – Cliente Satisfeito 121
2.4.6 – Cliente Insatisfeito 122
2.5 - Área de atuação 123
2.5.1 – Microrregião 123
2.5.2 – Macrorregião 125
2.6 – Concorrentes 126
2.6.1 – Concorrentes Diretos 126
2.6.2 – Concorrentes Indiretos 129
2.6.3 – Conclusão 130
2.7 – Legislação contratual 131
2.7.1 – Termos e Condições de Uso do Sistema BibliONtec 132
2.7.2 – Contrato de Prestação de Serviços 135
3 – DOCUMENTAÇÃO 139
3.1 – OMG 139
3.1.2 – Utilização no Sistema 141
3.2 – BPMN 141
3.2.1 – Aplicação no Sistema 143
3.2.2 – BPMN do administrador 145
3.2.3 – BPMN do aluno 146
3.2.4 – BPMN do funcionário (Bibliotecário) 148

3.2.5 – BPMN do professor 149
3.3 – Diagrama 150
3.3.1 – Diagrama do aluno 151
3.3.2 – Diagrama do administrador 152
3.3.3 – Diagrama do bibliotecário 154
3.3.4 – Diagrama do professor 155
3.3.5 – Diagrama de classe 157
3.4 – Aplicação *Web* 158
3.4.1 – Tela de *login* 159
3.4.2 – Tela Inicial do bibliotecário 164
3.4.3 – Área do bibliotecário 172
3.4.4 – Cadastro de usuários 178
3.4.5 – Cadastro de livros 183
3.4.6 – Cadastro de funcionários 191
3.4.7 – Empréstimo 194
3.4.8 – Acervo de livros 205
3.4.9 – Usuários cadastrados 216
3.4.10 – Funcionários cadastrados 221
3.4.11 – Tela Inicial administrador 226
3.4.12 – Área do administrador 228
3.4.13 – Administração de funcionários 229
3.4.14 – Administração de configurações 234
3.4.15 – Tela Inicial do professor 248
3.4.16 – Área de livros do professor 249
3.4.17 – Área do professor 250
3.4.18 – Indicar livros 253
3.4.19 – Tela Inicial do aluno 267
3.4.20 – Área do aluno 268
3.4.21 – Livros disponíveis aluno 269
3.4.22 - Comentários 271
Considerações Finais 277
Referências 280

INTRODUÇÃO

O avanço da tecnologia tem transformado a forma como a sociedade lida com tarefas cotidianas, promovendo agilidade, precisão e acessibilidade. Áreas que antes dependiam de registros manuais e processos burocráticos agora contam com soluções digitais que otimizam o tempo, reduzem falhas e melhoram a experiência dos usuários. Esse cenário se aplica também ao ambiente educacional, onde a gestão de bibliotecas ainda enfrenta desafios relacionados à organização do acervo, controle de empréstimos e comunicação com os usuários.

Na Escola Técnica Estadual (ETEC), identificou-se a necessidade de modernizar o sistema de funcionamento da biblioteca, uma vez que o modelo atual apresenta limitações tanto para os alunos quanto para os bibliotecários. A ausência de um sistema informatizado dificulta o controle eficiente dos livros, gera atrasos na devolução de exemplares e impede um acesso rápido às informações. Esses problemas impactam diretamente no desempenho do serviço e na experiência dos usuários.

Diante dessa realidade, surgiu o projeto BibliONtec — uma proposta de desenvolvimento de um sistema *web* voltado à modernização da biblioteca da ETEC. O sistema tem como objetivo principal facilitar o gerenciamento do acervo e agilizar os processos internos, permitindo que alunos e professores possam consultar os livros disponíveis de forma *online* e intuitiva. Para os bibliotecários, será oferecido um painel administrativo que permite o cadastro de exemplares, o controle de empréstimos, o monitoramento de prazos e a geração de relatórios, tudo integrado a um banco de dados estruturado.

A iniciativa também visa melhorar a comunicação entre os usuários e a biblioteca, além de reduzir erros administrativos e garantir uma gestão mais eficaz e transparente. A construção do BibliONtec será fundamentada em entrevistas com os usuários da biblioteca e nos dados coletados ao longo do desenvolvimento, buscando garantir que o sistema atenda às reais necessidades da escola. Dessa forma, espera-se que o projeto contribua para uma experiência mais moderna, eficiente e acessível no uso da biblioteca escolar.

1 – TECNOLOGIAS

É um produto da ciência e da engenharia que utiliza a aplicação do conhecido científico aos objetivos práticos da vida humana reunindo instrumentos, métodos e técnicas que propõe a resoluções de problemas (SIGNIFICADOS, 2025; LISBOA,2023).

A tecnologia tem evoluído nas últimas décadas modificando a maneira que se vive, trabalha e comunica, ela tem evoluído desde da era mercantil, quando as inovações ainda eram ligadas à agricultura. Com a Revolução Industrial surgiram as máquinas que automatizou a produção, mudando a economia. No século XX, com a evolução da tecnologia, a era da informação trouxe a *internet* e os computadores. Nos dias atuais, na era digital, vive-se em uma constante corrente de informações, com inovações como automação, *internet* das coisas e Inteligência Artificial (E–SAFER, 2025).

1.1 - LÓGICA DE PROGRAMAÇÃO

A lógica de programação tem um papel fundamental no processo de desenvolvimento de *softwares* e nas resoluções de problemas. Ela é o conjunto de regras e de técnicas que um programador utiliza na criação e desenvolvimento de programas de computador, aplicando a habilidade de pensar de forma lógica e separar em etapas o problema em questão, isso envolve a capacidade de decifrar padrões, organizar informações e achar soluções (CLEMENTE, 2025).

A lógica de programação envolve a criação de algoritmos que é justamente a sequência de passos para solucionar problemas que foram quebrados em partes menores, esses algoritmos bem projetados são mais eficientes e consomem menos recursos computacionais, o que é importante para o desenvolvimento de *softwares* (CLEMENTE, 2025).

1.2 - LINGUAGEM DE PROGRAMAÇÃO

As linguagens de programação são ferramentas importantes para comunicar-se com os computadores. Elas são formadas por comandos que, quando aplicados corretamente, realizam ações específicas. Pode-se dizer que ela começou nos anos 30, com os primeiros computadores elétricos. Em 1948, Konrad Zuse criou a

linguagem Plankalkül, embora na época não tivesse muita utilidade, então foi logo deixada de lado. Antes da programação com computadores, usavam-se cartões de papelão perfurados para criar códigos e instruções (INFOESCOLA, 2025).

Foi na década de 50 que as primeiras linguagens de programação modernas surgiram. Linguagens como FORTRAN *(Formula Translation* – Tradução de Fórmulas), LISP (*List Processing* – Processamento de Listas) e COBOL (*Common Business-Oriented Language* – Linguagem Orientada a Negócios Comum) foram criadas nesse período, além de ALGOL (*Algorithmic Language* – Linguagem Algorítima) 60. Cada uma delas tinha um propósito específico, e juntas ajudaram a definir os principais paradigmas da programação. Nos anos 70, linguagens como Simula, que introduziu o conceito de classes, e C, criada por Dennis Ritchie e Ken Thompson, tiveram uma enorme influência. O Prolog (*Programming in Logic* – Programação em Lógica), que trouxe o paradigma lógico, e o Pascal, uma linguagem focada no ensino, também marcaram essa época (INFOESCOLA, 2025).

Na década de 80, o C++ foi criado como uma versão mais dinâmica e flexível do C. Essa década também viu o surgimento de Perl, uma linguagem voltada para grandes volumes de dados e sobrecarga de informações. Nos anos 90, com a *internet* crescendo rapidamente, surgiram linguagens como Java, JavaScript e PHP (*Hypertext Preprocessor* – Pré-processador de Hipertexto). Java foi criada com a ideia de ser simples, enquanto PHP se tornou fundamental para o desenvolvimento de *sites* e aplicações *web*. Nessa época, o desenvolvimento da *web* ganhou uma grande força, e essas linguagens foram essenciais para isso (INFOESCOLA, 2025).

Hoje em dia, as linguagens de programação estão presentes em todos os dispositivos eletrônicos e *sites*. Com o tempo, elas se tornaram mais robustas e fáceis de aprender, e agora, as linguagens estão mais conectadas, o que facilita ainda mais a criação de programas e aplicativos. A programação passou por um processo de evolução bastante rápido, e hoje pode-se contar com ferramentas que tornam o desenvolvimento mais acessível e eficaz (INFOESCOLA, 2025).

Em resumo, as linguagens de programação se transformaram ao longo do tempo, se tornando mais poderosas, flexíveis e mais fáceis de usar. Elas são essenciais para o funcionamento de todos os dispositivos e da *internet*, e sua evolução

continua acelerando, criando novas oportunidades e desafios no mundo da tecnologia (INFOESCOLA, 2025).

1.2.1 – *Desktop*

O *desktop* é basicamente um computador de mesa, composto por um gabinete que guarda os componentes principais, como o processador, memória e placa de vídeo. Ao contrário do *notebook*, que já vem com monitor e teclado integrados, o *desktop* exige que essas peças sejam compradas separadamente, ou no máximo, pode-se escolher um modelo de gabinete e os componentes internos (QUEIROGA, 2022).

O termo *desktop* surgiu nos Estados Unidos em 1958, quando as pessoas começaram a usar para descrever objetos que poderiam ser colocados sobre uma mesa. Naquela época, os computadores eram enormes, ao ponto de ocupar até mesmo um a sala inteira (YUGE, 2025).

Para entender melhor o que é um *desktop*, é interessante saber a origem do nome. "*Desktop*" vem da tradução literal de "sobre a mesa". Ele pode se referir tanto à área de trabalho de um computador quanto ao tipo de computador projetado para ser colocado sobre uma mesa ou superfície similar. Aqui, está mostrando o *desktop* como um modelo de computador, que geralmente é montado em cima de uma mesa (YUGE, 2025).

Nos anos 60, a tecnologia começou a evoluir rapidamente, e os computadores começaram a diminuir de tamanho. Isso foi destacado em um artigo da revista *Computers and Automation*, onde o autor Rudy Stiefel questionava "Computadores grandes ou pequenos? Em que direção eles vão?" e mostrava como o mercado estava mudando. O *desktop*, que antes era algo muito grande, estava se tornando mais compacto e acessível (YUGE, 2025).

1.2.2 – *Mobile*

A plataforma *mobile* pode ser definida como a versão de *sites*, sistemas e aplicações pensada especificamente para dispositivos móveis, como *smartphones* e *tablets*. Seu desenvolvimento considera uma infraestrutura adaptada para telas menores e geralmente verticais, garantindo uma experiência de uso otimizada nesses aparelhos. Além disso, há dois tipos principais de aplicação *mobile*: os aplicativos

(APPS), que precisam ser baixados e instalados, e as versões *mobile* de *sites* ou sistemas, que funcionam diretamente no navegador, sem a necessidade de instalação. O termo "*mobile*", que significa "móvel" ou "portátil", começou a ser utilizado no contexto tecnológico em referência a dispositivos que pudessem ser facilmente transportados e utilizados em movimento. Essa designação se popularizou com a evolução dos celulares, especialmente a partir do lançamento do primeiro telefone celular comercial, o Motorola DynaTAC 8000X, em 1983. Embora fosse grande, pesado e com funções limitadas, ele marcou o início da era *mobile*. Durante os anos 90, os celulares começaram a se popularizar no Brasil, inicialmente com foco em chamadas telefônicas. Nessa mesma época, os *pagers* também eram usados para enviar mensagens curtas. As redes ainda eram analógicas, e os aparelhos, caros, sendo considerados símbolos de *status*. A transição para redes digitais no fim dos anos 90 e início dos anos 2000 trouxe celulares mais compactos e leves, além da popularização do envio de mensagens SMS (*Short Message Service* – Serviço de Mensagens Curtas), impulsionado por promoções das operadoras. No início dos anos 2000, surgiram os chamados "*feature phones*", que adicionaram funcionalidades como câmeras, toques polifônicos e acesso à *internet* via WAP *(Wireless Application Protocol* – Protocolo de Aplicação Sem Fio). Isso contribuiu para a maior acessibilidade e popularização dos celulares no país. No entanto, o grande marco na história dos dispositivos móveis foi o lançamento do iPhone pela Apple, em 2007. Com ele, foram introduzidos a tela sensível ao toque, o sistema operacional intuitivo e a App Store, que revolucionaram o conceito de *smartphone*. Logo após os dispositivos com sistema Android passaram a disputar o mercado, ampliando ainda mais o alcance da tecnologia *mobile*. A partir de 2010, os *smartphones* passaram a contar com recursos cada vez mais modernos, como câmeras de alta resolução, reconhecimento facial, carregamento sem fio e assistentes virtuais. A evolução das redes móveis, especialmente com a chegada do 4G, permitiu conexões mais rápidas, impulsionando o uso da *internet*, redes sociais, consumo de vídeos, *streaming* e outras funcionalidades. Com isso, o desenvolvimento *mobile* se tornou extremamente relevante para diversos segmentos. Hoje, os dispositivos móveis são parte fundamental da rotina das pessoas, sendo utilizados não apenas para comunicação, mas também para estudo, trabalho, lazer, socialização e consumo. Diante desse cenário, pensar em soluções *mobile* deixou de ser um diferencial e passou a ser uma

necessidade para qualquer projeto que deseje alcançar seu público de forma eficiente e moderna (TECNOSPEED 2021; AGENCIAFLOW, 2024).

1.2.3 – *Web*

A *web* (*World Wide Web*- Rede mundial de computadores*)*, é um sistema de informações interligadas que permite acessar conteúdos pela *internet*. Criada por Tim Berners-Lee em 1989, ela transformou a forma de compartilhar dados, possibilitando comunicação rápida e acesso a diversos conteúdos em qualquer lugar do mundo (PROFESSORJOSEHUMBERTO, 2025).

A *web* utiliza protocolos, sendo o HTTP (*Hypertext Transfer Protocol* – Protocolo de Transferência de Hipertexto) o principal. Ao digitar um endereço no navegador, ele envia uma solicitação ao servidor onde o *site* está hospedado. O servidor responde com os dados, que são exibidos na tela para o usuário interagir (PROFESSORJOSEHUMBERTO, 2025).

Os elementos básicos da *web* incluem navegadores, servidores e URLs (*Uniform Resource Locator* – Localizador Uniforme de Recursos). Os navegadores exibem as páginas, os servidores armazenam os dados e as URLs são os endereços que direcionam aos conteúdos desejados (PROFESSORJOSEHUMBERTO, 2025).

Além da *web* aberta, acessível a todos, existem a *web* profunda (conteúdos não indexados por buscadores) e a *web* semântica, que organiza melhor as informações ao dar significado aos dados (PROFESSORJOSEHUMBERTO, 2025).

A evolução da *internet* pode ser compreendida por meio de diferentes fases, conhecidas como *Web* 1.0, 2.0, 3.0 e, mais recentemente, *Web* 4.0. Cada uma dessas etapas foi marcada pela introdução de novas tecnologias e pela transformação na forma como os usuários interagem com o ambiente digital (CURVELO, 2022).

A *Web* 1.0 representava um modelo estático, onde as páginas funcionavam de maneira semelhante a documentos impressos, com pouca ou nenhuma interação por parte do usuário. Com o avanço da tecnologia, surgiu a *Web* 2.0, caracterizada pela valorização do conteúdo gerado pelo próprio usuário e pela adoção de recursos visuais mais sofisticados, como o uso das folhas de estilo em cascata (CSS – *Cascading Style Sheets*), que possibilitaram a criação de interfaces mais atrativas e dinâmicas (CURVELO, 2022).

Em seguida, a *Web* 3.0 trouxe consigo a descentralização e a automação dos processos, incorporando conceitos como o uso de dados estruturados, semântica e tecnologias como a *blockchain*. Nesse contexto, a inteligência artificial começou a ganhar maior protagonismo, tornando a experiência do usuário mais personalizada e eficiente (CURVELO, 2022).

Atualmente, com a ascensão da *Web* 4.0, o foco está na integração entre o mundo físico e digital, utilizando recursos avançados como a *Internet* das Coisas (IoT – *Internet of Things*), *Big Data* e sistemas autônomos, com o objetivo de proporcionar interações ainda mais inteligentes, automatizadas e conectadas (CURVELO, 2022).

1.3 – SERVIDORES

O termo servidor na área da computação surgiu a partir da teoria das filas, formalizada na notação de Kendall, e já era utilizado em registros como a RFC *(Request for Comments* – Solicitação de Comentários) 5 (1969), relacionada à ARPANET (*Advanced Research Projects Agency Network* – Rede da Agência de Projetos de Pesquisa Avançada) — precursora da *internet* — que diferenciava "servidor-*host*" e "usuário-*host*". À medida que as redes de computadores evoluíram, surgiu a necessidade de dedicar máquinas exclusivamente à prestação de serviços, consolidando o conceito moderno de servidor. Um servidor é um equipamento computacional composto por processadores, memórias*,* interfaces de comunicação, *softwares* e, em muitos casos, dispositivos de armazenamento como discos rígidos ou memórias SSD (*Solid State Drive* – Unidade de Estado Sólido). Ele pode ser físico ou virtual, funcionando local ou remotamente, com capacidade para processar aplicações, armazenar dados e fornecer serviços a outros dispositivos conectados em redes locais (LAN - *Local Area Network* – Rede de Área Local) ou remotas (WAN - *Wide Area Network* – Rede de Área Ampla). No modelo cliente-servidor, o servidor centraliza o acesso a informações, arquivos e recursos, atendendo às requisições de múltiplos dispositivos clientes. Essa estrutura permite o compartilhamento eficiente de dados, execução centralizada de aplicações e maior controle de segurança. Servidores instalados na *internet* geralmente ficam em *datacenters* e oferecem serviços como hospedagem de *sites*, transmissão de vídeos e músicas, envio de *e-mails*, entre outros. Esses servidores são configurados conforme as necessidades de desempenho, segurança e disponibilidade, podendo atuar em *clusters* para manter o

funcionamento contínuo mesmo em casos de falhas. A classificação dos servidores depende da sua função. Por exemplo: Servidor de aplicação: executa sistemas corporativos acessados por múltiplos usuários. Servidor de arquivos: gerencia o armazenamento e compartilhamento de dados. Servidor de banco de dados: realiza o processamento intenso de informações. Servidor de mídia: transmite conteúdos em *streaming*. Servidor de *e-mail*: gerencia mensagens eletrônicas. Outros: servidores de *backup*, FTP (*File Transfer Protocol* – Protocolo de Transferência de Arquivos), *proxy, web*, além de servidores especializados como NAS (*Network Attached Storage* – Armazenamento Conectado em Rede), DNS (*Domain Name System* – Sistema de Nomes de Domínio), Telnet (*Telecommunication Network* – Rede de Telecomunicações) e impressão. A escolha e configuração de um servidor dependem de fatores como o número de usuários, a carga de trabalho, o tipo de serviço oferecido e a possibilidade de escalabilidade. Com o avanço da tecnologia, os servidores passaram por uma evolução significativa. Entre 2010 e 2012, modelos como IBM M1, Dell 900/950 e HP G5 suportavam até dois processadores *quad-core*, 32 a 64 GB de RAM *(Random Access Memory* – Memória de Acesso Aleatório) e até 2 TB (*Terabyte)* de armazenamento. De 2012 a 2015, os modelos M2/M3 da IBM, Dell linha 10 e HP G6/G7 chegaram a processadores de seis núcleos e até 384 GB de RAM. Já entre 2015 e 2018, surgiram modelos ainda mais potentes, como a linha IBM M4, Dell linha 20 e HP G8, com até 12 núcleos por processador, 768 GB de RAM e até 12 TB de armazenamento. A partir de 2018 até os dias atuais, os modelos IBM M5, Dell linha 30/40 e HP G9/G10 apresentam suporte a processadores com até 56 núcleos, até 3 TB de RAM e até 18 TB de armazenamento, com foco em maior desempenho em leitura, gravação e processamento de dados (CONTROLE, 2025; MSSERVER, 2022).

1.3.1 – Locais

Um servidor local é um sistema de computação instalado fisicamente na própria empresa, utilizado para armazenar e gerenciar dados dentro de uma rede interna. Ele permite controle total sobre a infraestrutura e os arquivos armazenados, sendo ideal para organizações que priorizam segurança e acesso rápido às informações. Esse tipo de servidor é utilizado para hospedar aplicações, *sites* ou arquivos acessíveis apenas por dispositivos conectados à mesma rede, como em um

ambiente corporativo onde documentos são compartilhados entre os computadores da empresa, sem a necessidade de conexão com a *internet* (INFORTREND, 2025).

> O uso de um *Local Server* traz diversas vantagens para os desenvolvedores *web*. Primeiramente, ele permite que os desenvolvedores trabalhem em seus projetos mesmo sem uma conexão com a *internet*, o que é especialmente útil em locais onde a conexão é instável ou inexistente. Além disso, o *Local Server* oferece um ambiente isolado, onde os desenvolvedores podem testar suas aplicações sem afetar o ambiente de produção (NAPOLEON, 2025).

1.3.2 – *Web*

Um servidor *web* é um sistema responsável por disponibilizar conteúdos ou serviços por meio da *internet*. Sempre que um usuário digita um endereço (URL) no navegador ou clica em um *link*, é feita uma requisição ao servidor *web*, que processa essa solicitação e retorna a página ou o recurso solicitado (CARDOSO, 2024).

Esse mecanismo é fundamental para a hospedagem de *sites*, já que os servidores *web* são os responsáveis por manter o conteúdo acessível de forma contínua e eficiente. Eles são componentes essenciais da estrutura da *internet* moderna — sem eles, o acesso rápido e prático a informações *online* não seria possível (CARDOSO, 2024).

O funcionamento de um servidor *web* pode ser dividido em etapas principais: Recepção da requisição: o navegador envia uma solicitação HTTP ou HTTPS (*Hypertext Transfer Protocol Secure* – Protocolo de Transferência de Hipertexto Seguro) ao servidor, contendo a URL da página desejada e outros dados, como *cookies* e cabeçalhos. Interpretação da solicitação: o servidor analisa o pedido. Se for uma página estática, como um arquivo HTML (*Hypertext Markup Language* – Linguagem de Marcação de Hipertexto) ou imagem, ele apenas localiza e envia o arquivo. Geração de conteúdo dinâmico: no caso de páginas dinâmicas, o servidor pode executar *scripts* em linguagens como PHP, Python ou Java para gerar o conteúdo em tempo real. Retorno da resposta: por fim, o servidor envia a resposta ao navegador com os dados solicitados e informações adicionais nos cabeçalhos, como tipo de arquivo e instruções de *cache* (CARDOSO, 2024).

1.4 – NAVEGADORES

Os navegadores, também chamados de *web browsers*, são programas que permitem aos usuários acessar e interagir com documentos HTML disponíveis em

servidores *web*. São as ferramentas mais utilizadas para explorar a *World Wide Web*, que é a maior rede interligada de documentos em hipertexto (BRASILESCOLA, 2025).

O primeiro navegador foi desenvolvido por Tim Berners-Lee em 1990 e recebeu o nome de *World Wide Web*. Sua introdução no CERN (*Conseil Européen pour la Recherche Nucléaire* – Organização Europeia para Pesquisa Nuclear), em 1991, marcou o início do uso do hipertexto como forma de compartilhar informações. O crescimento da *Web*, no entanto, ocorreu de forma mais ampla com o lançamento do NCSA *(National Center for Supercomputing Applications* – Centro Nacional para Aplicações de Supercomputação) Mosaic em 1993, um navegador gráfico que teve versões para Unix, Macintosh e Windows. Posteriormente, Marc Andreessen, líder do projeto Mosaic, fundou a Netscape, que lançou o popular Netscape Navigator em 1994 (BRASILESCOLA, 2025).

A Microsoft, ao perceber o avanço da *Web*, adquiriu uma tecnologia da Spyglass Inc. e criou o Internet Explorer, iniciando assim a chamada “guerra dos navegadores”. Essa disputa pelo mercado levou as empresas a incluírem extensões proprietárias ao HTML, o que impactou negativamente os esforços de padronização da *Web*. Com a integração do Internet Explorer ao sistema operacional Windows e acordos com fabricantes, a Microsoft ampliou sua vantagem, levando ao declínio da Netscape, que em resposta tornou seu código aberto, dando origem ao projeto Mozilla (BRASILESCOLA, 2025).

Mesmo com dificuldades no desenvolvimento, o Mozilla evoluiu e resultou em uma suíte robusta de navegação, ainda que com menor participação no mercado. O navegador Opera, lançado em 1996, destacou-se por ser leve e rápido, com certa popularidade em dispositivos portáteis e em regiões da Europa. Já o Lynx, baseado apenas em texto, segue sendo utilizado em ambientes específicos (BRASILESCOLA, 2025).

No ambiente Mac (Macintosh), o Safari passou a ser o navegador padrão do sistema operacional da Apple. Baseado no motor KHTML (*KDE Hypertext Markup Language* – Linguagem de Marcação de Hipertexto da KDE), derivado do projeto de código aberto Konqueror, ele substituiu o Internet Explorer, cuja última versão para

Mac foi descontinuada após 2003. Entre os principais navegadores atuais, destacam-se o Internet Explorer (Microsoft), Opera e Mozilla Firefox (BRASILESCOLA, 2025).

1.5 - *SITES* ESTÁTICOS

Um *site* estático é composto por páginas com estrutura fixa, mantendo o mesmo visual e conteúdo sempre que acessado. Ele é geralmente desenvolvido com HTML, CSS e JavaScript, e apresenta uma construção mais simples e rápida em comparação aos *sites* dinâmicos, embora tenha funcionalidades limitadas. Apesar dessas limitações, é uma opção bastante comum para currículos, portfólios e páginas institucionais (HOSTINGER, 2024).

Esse tipo de *site* é formado por arquivos armazenados em servidores de hospedagem, os quais são renderizados diretamente nos navegadores dos visitantes. Ao acessar um *site* estático, o navegador solicita os arquivos ao servidor, que os envia exatamente como foram armazenados, sem alterações por parte de linguagens de programação do lado do servidor. Isso garante que todos os usuários visualizem o mesmo conteúdo, contribuindo para um desempenho mais rápido e estável (HOSTINGER, 2024).

Existem três abordagens comuns para a criação de *sites* estáticos: desenvolvê-los manualmente com HTML, CSS e JavaScript; utilizar geradores de *sites* estáticos que facilitam a produção de arquivos com base em modelos; ou criar o *site* por meio de ferramentas visuais conhecidas como criadores de *sites*, voltadas especialmente para iniciantes (HOSTINGER, 2024).

Diversos *sites* utilizam essa abordagem. O *site* do *framework* Bootstrap, por exemplo, é estático, simples e com foco em desempenho. Já o *site* da TwitchCon, criado com Jekyll, apresenta um *layout* básico com foco em navegação e informações diretas. O estúdio Collins mostra que é possível criar um *site* estático visualmente atrativo, com recursos como *banners*, vídeos e animações leves. O desenvolvedor Camilo Holguin usa um *site* estático em Gatsby para exibir seu portfólio profissional em uma única página, com *links* para redes sociais. Por fim, o *site* da linguagem Prism também é estático, com foco em desempenho e simplicidade, oferecendo uma navegação rápida entre conteúdos técnicos (HOSTINGER, 2024).

1.6 – HTML

O HTML (*Hypertext Markup Language* - Linguagem de Marcação de Hipertexto) é uma linguagem de marcação de Hipertexto. Ele é o componente básico da *web*, estabelecendo a estrutura básica dos *websites*. Sem ele, os *sites* não exibiriam elementos como textos, ou imagens (MARQUES, 2024).

O HTML foi inventado por Tim Berners-Lee na Suíça, sua primeira versão sendo lançada em 1991. Ele surgiu como uma ideia de um projeto para interligar instituições de pesquisa e compartilhar documentos com mais facilidade. Em 1992, a biblioteca WWW (*World Wide Web* – Rede Mundial de Documentos Interligados) foi liberada e, junto do HTML, impulsionou mundialmente o uso da *WEB* (MARQUES, 2024).

Em sua primeira versão, conhecida como HTML 1.0, foi lançada em 1991 e incluía apenas funções básicas como títulos, parágrafos, *links* e imagens em preto e branco. Em 1995, foi atualizado para o HTML 2.0, trazendo o suporte a tabelas, formulários e *framesets*. Em 1997, foi lançado o HTML 3.0, que já poderia suportar o CSS (*Cascading Style Sheets* - Folhas de Estilo em Cascata), além de elementos como vídeo e áudio. Em 1999, foi lançado o HTML 4.0, aprimorando elementos e trazendo novos, como tabelas avançadas, *frames*, suporte para *scripts* como o Javascript. Em 2014, foi lançado o HTML5, trazendo muitos recursos novos como elementos semânticos, suporte para mídia audiovisual, geolocalização e armazenamento local. Desde então, o HTML5 se tornou a versão mais utilizada e tem suporte em quase todos os navegadores, além de ainda receber atualizações (MARQUES, 2024).

1.7 – CSS

O CSS (*Cascading Style Sheets* - Folhas de Estilo em Cascata) é uma linguagem de estilo utilizada para definir e controlar a apresentação visual de documentos criados com HTML. Em vez de embutir as regras de formatação diretamente no conteúdo, o CSS permite separar a estrutura da estética, possibilitando maior flexibilidade e padronização na exibição de cores, fontes, *layouts* e demais elementos de uma página *web* (SANTANA, 2025).

> O CSS teve suas raízes no surgimento da *World Wide Web*. Tim Berners-Lee, o inventor da *web*, lançou a primeira página da *web* em 1991. No entanto, as páginas iniciais eram simples, sem muita preocupação com a estilização. Foi somente em 1996 que o *World Wide Web Consortium* (W3C) lançou o CSS1, o primeiro padrão oficial do CSS. Esse foi um marco significativo que definiu como os estilos poderiam ser aplicados aos documentos HTML. O CSS1 permitiu a separação entre a estrutura do documento e seu estilo, revolucionando a forma como as páginas eram projetadas (FERNANDO, 2025).

Com a chegada do CSS2, houveram algumas mudanças relativamente importantes, introduzindo novos recursos como melhorias no posicionamento, seletores avançados, entre outros. Contudo, ainda havia limitações, principalmente no quesito da criação de *layouts* complexos. O CSS3, no entanto, trouxe mudanças ainda maiores, e corrigindo problemas e limitações. As principais funcionalidades adicionadas foram: Flexbox, que permitia a criação de *layouts* mais dinâmicos; Grid Layout, que proporcionava controle preciso e a criação de *layouts* bidimensionais; transições e animações; transformações 2D e 3D; e Media Queries, essencial para o desenvolvimento responsivo (REIS, 2023).

1.8 – *SITES* DINÂMICOS

> O *site* dinâmico é uma página da *web* controlada por um servidor de aplicação com tecnologias que rodam ao lado do servidor. Geralmente, recebem uma entrada de dados dos visitantes, os processam no servidor e devolvem como informação. As modificações que podem ser realizadas são inúmeras e ilimitadas. Ou seja, você terá total liberdade para mudar a funcionalidade e o visual das páginas (KBRTEC, 2019).

Sites dinâmicos são mais atrativos, com mais ferramentas, interações e uma maior autonomia. Eles possuem um painel de gerenciamento, que permite mudanças e gestão sem que seja necessária a alteração do código fonte. Sua principal desvantagem, se comparado aos *sites* estáticos, é o seu custo, do qual normalmente é mais elevado (KBRTEC, 2019).

1.9 – JAVASCRIPT

O JavaScript é uma linguagem usada para fazer páginas interativas, que melhorem a experiência do usuário, usando animações, elementos interativos, entre outras coisas. É uma das tecnologias principais da *World Wide Web* (AMAZON, 2025).

> O JS ou JavaScript é uma linguagem de programação de alto-nível, criada no meio da década de 90, mais precisamente em 1996 pelo lendário programador Brendan Eich que, além de criar o JavaScript, foi também um dos fundadores da Mozilla Corporation. Ele criou a linguagem quando ainda trabalhava na Netscape Communications Corporation, originalmente criado para funcionar no navegador Netscape Navigator, seu objetivo era facilitar processos dentro de páginas *web*, tornado a programação de animações e alertas muito mais simples. Pouco tempo depois, cerca de 1 ano depois do seu lançamento, a Microsoft começou a aceitar esta

> linguagem em seu navegador da época, o que foi um dos grandes impulsionadores para consolidar o que é Javascript como uma das principais linguagens de programação dos dias de hoje (HOSTINGER, 2025).

Em 1997, a linguagem foi padronizada através do ECMAScript (ES), o que estabeleceu as bases para sua evolução contínua. Desde então, JavaScript passou por várias atualizações significativas: As primeiras versões do ECMAScript (ES1, ES2 e ES3) estabeleceram os fundamentos da linguagem. Em 2009, o ECMAScript 5 (ES5) introduziu melhorias importantes como métodos mais sofisticados para manipulação de objetos e *arrays*, aumentando a robustez da linguagem. A grande revolução ocorreu com o ECMAScript 6 (ES2015), que trouxe inovações como classes, *arrow functions*, *let/const*, promessas, módulos e muito mais, tornando JavaScript muito mais poderoso e moderno para desenvolvimento tanto no *front-end* quanto no *back-end*. Versões subsequentes (como ES2016, ES2017, etc.) continuam a adicionar recursos que aprimoram a sintaxe, desempenho e capacidades da linguagem (HANASHIRO, 2024).

1.10 – PHP

PHP é uma linguagem de *script* voltada para o lado do servidor, capaz de realizar diversas funções de *back-end*, como o processamento de formulários, manipulação de arquivos no servidor, atualização de bancos de dados, entre outras tarefas. Embora seja uma linguagem de uso geral, seu principal foco está no desenvolvimento *web*, principalmente por sua facilidade de integração com arquivos HTML (HOSTINGER, 2024).

Com PHP, é possível ocultar o código-fonte do usuário final, protegendo o conteúdo original. Também facilita a reutilização de trechos HTML, evitando repetição de código ao permitir a inclusão de *scripts* PHP onde for necessário (HOSTINGER, 2024).

Um exemplo prático pode ser visto no próprio Facebook, onde URLs como "facebook.com/home.php" indicam o uso da linguagem na construção da página, mesclando comandos PHP com elementos HTML (HOSTINGER, 2024).

Criado inicialmente como um conjunto de *scripts* CGI (*Common Gateway Interface* – Interface de *Gateway* Comum) escritos em linguagem C, o PHP nasceu da necessidade de monitorar acessos ao *site* pessoal de seu criador, Rasmus Lerdorf.

Com o tempo, ele foi aprimorando os recursos e o sucesso foi tanto que acabou se tornando uma linguagem de programação completa, adotada por inúmeros *sites* e aplicações (HOSTINGER, 2024).

Em 1997, o PHP passou por uma grande reformulação, ganhando o nome de PHP Hypertext Preprocessor e incorporando recursos de orientação a objetos. Desde então, novas versões foram lançadas, como a 4.0, que trouxe avanços, mas também apresentou limitações corrigidas na versão 5.0 — ainda bastante utilizada. Atualmente, a linguagem está na versão 7.2, mas a adoção de versões pode variar conforme o servidor de hospedagem (HOSTINGER, 2024).

Para aplicar o PHP em projetos, é essencial compreender sua lógica e estrutura, dando os primeiros passos no desenvolvimento de sistemas *web* com base em suas funcionalidades (HOSTINGER, 2024).

1.11 – NODE.JS

Node.js (Node JavaScript) é um ambiente de execução de código JavaScript no lado do servidor, baseado no motor V8 do Google Chrome. Foi criado por Ryan Dahl em 2009 com o objetivo de oferecer uma forma mais eficiente de lidar com conexões simultâneas, superando limitações de servidores tradicionais como o Apache HTTP *Server*. É uma ferramenta de código aberto e multiplataforma (compatível com Windows, Linux e macOS), amplamente utilizada por empresas como Netflix, PayPal, LinkedIn, entre outras (MARCHIORI, 2022).

Seu diferencial está no modelo assíncrono e orientado a eventos, que evita o bloqueio do servidor durante a execução de tarefas. Com isso, mesmo com apenas uma *thread* principal, o Node.js consegue atender múltiplas requisições simultaneamente, tornando-o ideal para aplicações em tempo real, como *chats*, APIs (*Application Programming Interfaces* - Interface de Programação de Aplicações), microsserviços e automações. No entanto, seu uso não é recomendado em aplicações extremamente complexas ou que dependem fortemente de SGBDs relacionais (Sistemas de Gerenciamento de Banco de Dados) como MySQL, PostgreSQL ou SQL *Server* (*Structured Query Language* – Linguagem de Consulta Estruturada), pois esse tipo de estrutura pode comprometer a performance (MARCHIORI, 2022).

Um dos grandes pilares do Node.js é o NPM (*Node Package Manager* - Gerenciador de Pacotes Nodes), seu gerenciador oficial de pacotes, que oferece milhares de bibliotecas reutilizáveis. Com ele, é possível instalar e gerenciar dependências de forma rápida e prática. Dentre os módulos mais populares estão o *Express* (*Express.js*) *framework* para criação de servidores e rotas, MongoDB – integração com banco de dados NoSQL –, Nodemailer envio de *e-mails* e Lodash biblioteca com funções utilitárias. Para manter a compatibilidade entre diferentes projetos, existe o NVM (*Node Version Manager* – Gerenciador de Versões para Node), ferramenta que permite instalar e alternar entre várias versões do Node.js com facilidade (MARCHIORI, 2022).

Com uma comunidade ativa, inúmeros eventos e constantes atualizações, o Node.js se consolidou como uma tecnologia moderna e poderosa dentro do ecossistema JavaScript, oferecendo agilidade, desempenho e flexibilidade para desenvolvedores e desenvolvedoras em diversas áreas da programação (MARCHIORI, 2022).

1.12 – VISUAL STUDIO CODE

O Visual Studio Code, ou simplesmente VS (Visual Studio) Code, é um editor de código-fonte criado pela Microsoft e lançado em 2015. Mesmo sendo relativamente recente, conquistou grande popularidade entre os desenvolvedores devido à sua versatilidade, leveza e eficiência. Ele é compatível com os sistemas operacionais Windows, macOS e Linux, e é uma ferramenta *open source*, ou seja, com código aberto, permitindo que a comunidade colabore com melhorias constantes. Voltado especialmente para programadores, o VS Code facilita a escrita, edição e gestão de códigos, oferecendo recursos que otimizam o processo de desenvolvimento, desde a codificação até a fase de testes. Entre os seus principais diferenciais estão (REMESSAONLINE, 2021):

- Interface intuitiva e personalizável: Permite que o usuário adapte o ambiente de acordo com suas preferências, com diversos temas e configurações (REMESSAONLINE, 2021).
- Suporte a diversas linguagens: O editor suporta as principais linguagens de programação como JavaScript, HTML, CSS, PHP, Java, Python, C++, C#, SQL

(*Structured Query Language* – Linguagem de Consulta Estruturada), entre outras (REMESSAONLINE, 2021).

- IntelliSense: Recurso inteligente que sugere *autocompletes* de código com base na linguagem utilizada, variáveis, funções e módulos importados, agilizando e facilitando a escrita (REMESSAONLINE, 2021).
- Depuração Integrada: Permite ao programador depurar seu código diretamente no editor, utilizando pontos de interrupção, pilhas de chamadas e console interativo para identificar e corrigir erros (REMESSAONLINE, 2021).
- Integração com o Git: Facilita o controle de versões ao exibir alterações, permitir comparações e realizar *commits* diretamente pela interface do editor (REMESSAONLINE, 2021).
- Integração com o Azure: Possibilita implantar e hospedar aplicações, gerenciar bancos de dados e utilizar recursos em nuvem sem sair do editor (REMESSAONLINE, 2021).

Além dessas funcionalidades nativas, o VS Code se destaca por seu vasto ecossistema de extensões, que ampliam ainda mais sua capacidade e produtividade. Algumas das mais utilizadas são (CAVALCANTI, 2023):

- *Live Server*: Cria um servidor local para visualizar atualizações em tempo real em projetos HTML, CSS e JavaScript, sem a necessidade de recarregar manualmente o navegador (CAVALCANTI, 2023).
- Visual Studio IntelliCode: Utiliza inteligência artificial para oferecer sugestões de código baseadas no contexto do projeto (CAVALCANTI, 2023).
- Beautify: Organiza e padroniza automaticamente a indentação e a estrutura de códigos HTML, CSS, JavaScript, entre outros (CAVALCANTI, 2023).
- *Bookmarks*: Permite marcar linhas ou trechos importantes do código para facilitar revisões posteriores (CAVALCANTI, 2023).
- *Auto Import*: Localiza e insere automaticamente importações necessárias no código (CAVALCANTI, 2023).
- VSCode *Icons*: Adiciona ícones visuais que identificam os tipos de arquivos e tecnologias usadas, melhorando a organização (CAVALCANTI, 2023).

- GitLens: Oferece uma visualização avançada do histórico de alterações do Git, mostrando detalhes como autor, data e tipo de modificação (CAVALCANTI, 2023).
- *Brackets Pair Colorizer*: Destaca pares de colchetes com cores diferentes, facilitando a leitura de códigos mais complexos (CAVALCANTI, 2023).

O Visual Studio Code, portanto, é mais do que um simples editor: trata-se de uma plataforma robusta, personalizável e eficiente que acompanha o programador em todas as etapas do desenvolvimento, contribuindo para a qualidade, clareza e produtividade dos projetos de *software* (REMESSAONLINE, 2021; CAVALCANTI, 2023).

1.13 - GITHUB

O GitHub é uma plataforma baseada na nuvem que permite armazenar, compartilhar e colaborar no desenvolvimento de códigos-fonte. Criado por Chris Wanstrath, J. Hyett, Tom Preston-Werner e Scott Chacon, e lançado oficialmente em 2008, foi desenvolvido com Ruby on Rails e se consolidou como uma das principais ferramentas para controle de versão e trabalho em equipe. Seu funcionamento é baseado no Git, um sistema de versionamento que possibilita acompanhar modificações no código ao longo do tempo, permitindo que diversos colaboradores atuem simultaneamente sem interferir diretamente no trabalho uns dos outros. Além de facilitar o gerenciamento de alterações, o GitHub permite que usuários compartilhem seus projetos, recebam sugestões e colaborem de forma organizada. Sua interface amigável e acessível atrai tanto iniciantes quanto profissionais experientes, tornando-o uma escolha popular entre desenvolvedores e grandes empresas. Com mais de 100 milhões de repositórios — a maioria de código aberto — a plataforma cresceu rapidamente: em 2009 já contava com 100 mil usuários e 90 mil repositórios, atingindo 1 milhão de repositórios em 2010, e 2 milhões no ano seguinte. Em 2015, abriu seu primeiro escritório fora dos EUA, no Japão, e foi avaliada em cerca de 2 bilhões de dólares após um investimento significativo. Em 2018, o GitHub foi adquirido pela Microsoft por US$ 7,5 bilhões e passou a oferecer pacotes educacionais gratuitos para instituições de ensino. Hoje, é reconhecido como uma das principais ferramentas para desenvolvimento colaborativo e versionamento de projetos, oferecendo recursos robustos de integração, revisão de código e

gerenciamento de equipe em um único ambiente acessível (DOCSGITHUB, 2025; HOSTINGER, 2024).

1.14 – BANCO DE DADOS

> Um banco de dados é uma coleção organizada de informações - ou dados - estruturadas, normalmente armazenadas eletronicamente em um sistema de computador. Um banco de dados é geralmente controlado por um sistema de gerenciamento de banco de dados (DBMS - *Database Management System* - Sistema de Gerenciamento de Banco de Dados). Juntos, os dados e o DBMS, juntamente com os aplicativos associados a eles, são chamados de sistema de banco de dados, geralmente abreviados para apenas banco de dados (ORACLE, 2020).

A história dos bancos de dados começa nas décadas de 1960 e 1970, quando a IBM (*International Business Machines*) identificou os altos custos de armazenar dados manualmente e iniciou pesquisas para otimizar esse processo. Durante esse período, Ted Codd, pesquisador da IBM, publicou um artigo pioneiro sobre bancos de dados relacionais, propondo o uso de álgebra relacional para que usuários sem conhecimentos técnicos pudessem armazenar e recuperar informações de maneira eficiente. Essa abordagem incentivou a criação do grupo de pesquisa System R, que mais tarde evoluiu para o DB2 (*Database* 2 – Base de Dados 2), sistema de banco de dados da própria IBM (UFRN, 2025).

Simultaneamente, outras empresas também investiram nesses novos conceitos. Por exemplo, em 1976, a Honeywell Information Systems Inc lançou o primeiro sistema comercial de banco de dados, baseado em ideias semelhantes às propostas pela IBM. Logo depois, surgiram sistemas como o Oracle 2, desenvolvido pela Oracle, e o SQL/DS (*Structured Query Language/Data System* – Linguagem de Consulta Estruturada/Sistema de Dados) da IBM, que introduziu a linguagem SQL (*Structured Query Language*) – uma ferramenta essencial para a manipulação e consulta de dados (UFRN, 2025).

Na década de 1990, vários sistemas relacionais ganharam destaque, incluindo DBase III, Paradox, SQL *Server* e MySQL, ampliando as possibilidades de armazenamento e gerenciamento de informações. Paralelamente, o conceito de banco de dados orientado a objetos (BDOO) emergiu para atender necessidades específicas em áreas como medicina, multimídia e física avançada, impulsionando também o desenvolvimento de linguagens de programação orientadas a objetos, como o C++ (UFRN, 2025).

Mais recentemente, com o crescimento exponencial da *internet* e o aumento da geração de dados por meio de serviços de busca e redes sociais, surgiram os bancos de dados NoSQL (*Not Only Structured Query Language* – Não apenas Linguagem de Consulta Estruturada), que priorizam a escalabilidade, a replicação e a flexibilidade, mesmo que isso signifique uma menor ênfase na consistência dos dados (UFRN, 2025).

1.14.1 – Modelagem

A modelagem de dados é um processo essencial na estruturação de bancos de dados, sendo responsável por representar graficamente o fluxo das informações. Quando um sistema ou banco de dados está em fase de desenvolvimento, é necessário criar um esquema visual que ilustre como os dados serão inseridos, processados e consultados. Esse modelo é utilizado como base para definir as características dos formatos, das estruturas e das funcionalidades do sistema de armazenamento de dados, garantindo que ele atenda às necessidades específicas do projeto. Após a implementação, esse modelo serve como documentação técnica e também como referência para a compreensão de como o banco foi estruturado (SAP, 2025).

O resultado da modelagem é uma representação das relações entre os componentes do banco, funcionando como um guia para o uso adequado dos dados. Esse processo é essencial tanto no desenvolvimento de sistemas quanto nas análises de dados, pois padroniza e organiza as informações, permitindo que diferentes aplicações compartilhem os mesmos dados de forma eficaz (SAP, 2025).

A modelagem de dados proporciona um banco de dados mais limpo e eficiente, eliminando informações duplicadas, reduzindo o espaço de armazenamento necessário e facilitando a recuperação dos dados. Ela também assegura que todos os sistemas envolvidos tenham acesso a uma versão unificada da informação, o que é fundamental para a eficiência dos processos e o cumprimento de normas regulatórias (SAP, 2025).

Antes da criação de um sistema, é necessário documentar como ele funcionará, tanto em relação às regras de negócio quanto à forma como os dados circularão e serão armazenados. A modelagem garante essa visão e direciona os

desenvolvedores, permitindo que o sistema final entregue os resultados esperados, desde que os processos sejam corretamente seguidos (SAP, 2025).

Com o crescimento constante do volume de dados e da quantidade de usuários, as empresas precisam transformar esses dados brutos em informações úteis. Nesse contexto, a modelagem de dados ajuda a preparar as informações para análise, selecionando o que será utilizado, definindo medidas e dimensões, além de possibilitar a criação de fórmulas e hierarquias para facilitar a visualização gráfica e a tomada de decisão (SAP, 2025).

Existem diferentes formas de estruturar modelos de dados, sendo as mais comuns: relacional, dimensional e entidade-relacionamento (E-R). O modelo relacional é amplamente utilizado e organiza os dados em tabelas com linhas e colunas. Já o modelo dimensional é mais flexível e organiza os dados de acordo com seu uso no negócio, favorecendo análises rápidas. O modelo entidade relacionamento, por sua vez, representa graficamente os elementos do banco (as entidades) e suas relações, sendo muito usado para criar a estrutura de bancos relacionais (SAP, 2025).

O desenvolvimento da modelagem pode ser pensado em três etapas principais: uma visão mais ampla e geral da estrutura, um detalhamento intermediário e, por fim, a parte prática e específica do banco. O primeiro nível, mais abstrato, ajuda a visualizar o sistema como um todo, enquanto o segundo descreve de forma mais detalhada o fluxo de dados. A terceira etapa mostra exatamente como o banco será implementado, com base na tecnologia escolhida (SAP, 2025).

A criação desses modelos geralmente segue uma lógica que começa com ideias e representações gráficas. As ferramentas atuais facilitam esse processo com recursos visuais que permitem montar os modelos de forma intuitiva. Com essas ferramentas, é possível arrastar e soltar componentes, estabelecer conexões e montar visualizações que combinam diferentes conjuntos de dados, facilitando a análise e o desenvolvimento de sistemas (SAP, 2025).

A modelagem é fundamental em qualquer sistema, seja comercial, educacional, de entretenimento ou outro. Ela é necessária para garantir que os dados estejam bem estruturados e sejam compatíveis entre si, o que é essencial

principalmente em sistemas que lidam com informações de várias fontes, como em um *data warehouse*. Nesses casos, a modelagem permite transformar formatos diferentes em algo padronizado e útil para análises e geração de relatórios (SAP, 2025).

Desde os primeiros sistemas computacionais, a modelagem de dados tem sido uma etapa indispensável. Apesar de o termo ter se popularizado com o avanço dos bancos de dados na década de 1960, a prática de planejar e estruturar dados sempre existiu. Atualmente, com o crescimento do volume de dados, o surgimento de novas fontes como sensores, redes sociais e dispositivos inteligentes, a modelagem se tornou ainda mais importante (SAP, 2025).

Com as inovações tecnológicas e o aumento da conectividade, a modelagem de dados também evoluiu. O uso da computação em nuvem e de bancos de dados mais modernos permite uma maior escalabilidade e agilidade na manipulação dos dados. Hoje, muitos sistemas utilizam dados armazenados na memória, em vez do disco, e adotam estruturas mais otimizadas, o que melhora o desempenho e a eficiência (SAP, 2025).

O futuro da modelagem está ligado à necessidade de adaptar-se à diversidade e ao volume crescente de dados. Novas ferramentas e abordagens devem continuar surgindo para tornar o processo mais acessível, colaborativo e eficiente, garantindo que profissionais de diferentes áreas possam trabalhar com dados de forma integrada e produtiva (SAP, 2025).

1.14.2 – Modelo conceitual

> É o modelo de mais alto nível, ou seja, que está mais próximo da realidade dos usuários. O nível conceitual é desenvolvido com alto nível de abstração, a partir dos requisitos do sistema, extraídos na fase de levantamento de requisitos. Esse modelo pode ser elaborado por meio de dois diagramas: Diagrama de Entidade e Relacionamento e/ou o Diagrama de Classes (SPACEPROGRAMMER, 2025).

No modelo conceitual, os dados são organizados em entidades. Ele permite que os usuários entendam e comuniquem os requisitos, sem se preocupar com detalhes. Ele também é a base para o modelo lógico e físico, pois dá uma estrutura clara (SALES, 2024).

1.14.3 – Modelo lógico

> O modelo lógico de banco de dados é uma representação abstrata da estrutura de um banco de dados. Ele descreve como os dados estão organizados, relacionados e armazenados no banco de dados, sem se preocupar com os detalhes físicos de implementação. O modelo lógico fornece uma visão conceitual dos dados, permitindo que os desenvolvedores projetem e criem sistemas de banco de dados eficientes e flexíveis (FEDERALCUBATAO, 2024).

O modelo lógico ainda não é focado em detalhes da implementação, mas captura as necessidades de forma clara e compreensível (ALVES, 2024).

1.14.4 – Modelo físico

> O modelo físico traduz o modelo lógico em uma estrutura concreta e detalhada de dados que será implementada no sistema de banco de dados. Aqui, entidades são transformadas em tabelas, atributos em colunas com tipos de dados específicos e relacionamentos são estabelecidos através de chaves estrangeiras. Aspectos como índices, restrições de integridade e otimizações de desempenho são considerados neste estágio (ALVES, 2024).

Em resumo, o modelo físico traduz o modelo lógico, para facilitar a implementação em termos concretos, de forma a garantir que os requisitos sejam atendidos (ALVES, 2024).

1.15 – BR MODELO

O brModelo é um *software* de código aberto destinado à modelagem de bancos de dados relacionais, desenvolvido em 2005 pelo Grupo de Pesquisa em Banco de Dados da Universidade Federal de Santa Catarina (GPD/UFSC), em parceria com a UNIVAG (MT – Mato Grosso), como parte de um trabalho de conclusão do curso de especialização em banco de dados. A orientação foi do Professor Dr. Ronaldo dos Santos Mello, e a ferramenta foi idealizada com base na metodologia apresentada por Carlos A. Heuser, no livro “Projeto de Banco de Dados”. Seu principal objetivo é oferecer uma solução gratuita e eficiente para o ensino e prática de modelagem de dados, especialmente no contexto acadêmico. Um dos grandes diferenciais do brModelo em relação a outras ferramentas é o suporte às três etapas clássicas do projeto de banco de dados: modelagem conceitual, lógica e física. Além disso, ele oferece interação direta com o projetista durante a modelagem lógica e total suporte aos conceitos do modelo entidade-relacionamento estendido (EER - *Extended Entity-Relationship*), conforme abordado nas principais literaturas brasileiras da área. Isso contribuiu para sua ampla aceitação e popularização, especialmente em instituições de ensino técnico e superior. Em junho de 2006, foi lançada a versão 2.0

da ferramenta. Na época, especulava se que, em cerca de uma década, a modelagem relacional poderia ser substituída por abordagens orientadas a objetos, nas quais o modelo de classes seria persistido diretamente em sistemas gerenciadores de banco de dados (SGBDs), dispensando uma etapa exclusiva de modelagem de dados. No entanto, essa expectativa não se concretizou, e os bancos de dados relacionais continuaram predominantes. Em 2015, mesmo após dez anos do lançamento inicial, a demanda por conhecimento na área de modelagem relacional permaneceu significativa, o que reforçou a importância da ferramenta. Nesse contexto, surgiu a necessidade de modernizar o brModelo. Assim, foi desenvolvida uma nova versão, também gratuita e de código aberto, porém construída com uma linguagem de programação mais atual, voltada para ambientes colaborativos. A versão beta 0.1 foi apresentada durante a Escola Regional de Banco de Dados de 2017, realizada em Passo Fundo (RS), onde foram coletadas sugestões de melhorias com o público presente. Atualmente, o brModelo continua sendo amplamente utilizado no Brasil e em outros países, consolidando-se como uma das principais ferramentas para o ensino de modelagem de dados relacionais (MUNDODACOMPUTACAOINTEGRAL, 2021; CANDIDO, 2025).

1.16 – XAMPP

O XAMPP (X (multiplataforma), Apache, MySQL, PHP, Perl) é um pacote que reúne alguns dos principais servidores de código aberto utilizados no desenvolvimento *web*, como o servidor FTP, o banco de dados MySQL e o servidor Apache, além de oferecer suporte às linguagens PHP e Perl. Esse pacote possibilita a execução de sistemas como WordPress e Drupal em ambiente local, o que contribui para um desenvolvimento mais rápido e eficiente. Como os arquivos ficam armazenados em uma rede local, o acesso a eles ocorre de forma imediata. Segundo dados do *site* SourceForge, o XAMPP é baixado aproximadamente 600 mil vezes por mês (HIGA, 2012).

> O XAMPP foi criado por Kai ‘Oswald’ Seidler e Kay Vogelgesang da Apache Friends em 2002. Inicialmente, o XAMPP era destinado apenas ao sistema operacional Windows, mas posteriormente foi expandido para incluir suporte para Linux e macOS (EXPERTEMTI, 2024).

1.17 – MYSQL

> O MySQL é o sistema de gerenciamento de banco de dados de código aberto mais conhecido no mundo. Os bancos de dados são os repositórios de dados essenciais para todas as aplicações de *software.* Por exemplo, sempre que alguém realiza uma pesquisa na *Web*, faz *login* em uma conta ou conclui uma transação, um banco de dados armazena as informações para que elas possam ser acessadas no futuro. O MySQL se destaca nessa tarefa. SQL, que significa linguagem de consulta estruturada em inglês, é uma linguagem de programação usada para recuperar, atualizar, excluir e manipular dados em bancos de dados relacionais. Oficialmente, "SQL" deve ser lido como uma sigla, mas é comumente usada como um acrônimo. Como o nome sugere, o MySQL é um banco de dados relacional baseado em SQL projetado para armazenar e gerenciar dados estruturados (ERICKSON, 2024).

O MySQL é um sistema de gerenciamento de banco de dados relacional que teve sua origem fundamentada no modelo relacional descrito por Edgar F. Codd. Desenvolvido pela empresa sueca MySQL AB, fundada por David Axmark, Allan Larsson e Michael Widenius, o MySQL foi lançado em 1995. O nome foi formado a partir da junção de *“My”*, em homenagem à filha de Michael Widenius, e “SQL” (*Structured Query Language* – Linguagem de Consulta Estruturada), que é a linguagem utilizada para gerenciar e consultar dados em bancos de dados. Em 2008, a MySQL AB foi adquirida pela Sun Microsystems, e posteriormente, em 2010, a Oracle Corporation passou a deter o MySQL com a aquisição da Sun Microsystems (GOOGLE, 2025).

1.17.1 – MariaDB

> O MariaDB é um sistema de gerenciamento de banco de dados relacional (RDBMS, *Relational Database Management System)* gratuito e de código aberto. Ela foi criada pelos desenvolvedores originais do MySQL em meio às preocupações de que o MySQL seria comercializado depois que a Oracle o adquiriu em 2009 (PURESTORAGE, 2025).

O MariaDB foi criado por Michael “Monty” Widenius, também fundador da MySQL AB e lançado em 2009. Surgiu como um *fork* do MySQL. Principais versões do MariaDB (PURESTORAGE, 2025):

- MariaDB 5.5: Representou as primeiras versões do MariaDB, baseadas no MySQL, mas com aprimoramentos de desempenho e escalabilidade que serviram de alicerce para futuras evoluções (MARIADB, 2025).
- MariaDB 10.0: Marcou o início da série 10.x, trazendo novos recursos, melhores práticas de gerenciamento e uma arquitetura mais robusta, diferenciando-se das versões anteriores (MARIADB, 2025).

- MariaDB 10.1: Introduziu melhorias adicionais em segurança, replicação e desempenho, adequando-se ainda mais às demandas de aplicações empresariais (MARIADB, 2025).
- MariaDB 10.2: Ampliou os recursos de armazenamento, melhorou a performance na manipulação de dados e refinou mecanismos de replicação, garantindo maior escalabilidade (MARIADB, 2025).
- MariaDB 10.3: Focou na otimização de consultas e gerenciamento de dados, adicionando funcionalidades como compressão de dados e outras melhorias que aumentaram a eficiência do sistema (MARIADB, 2025).
- MariaDB 10.4, 10.5 e 10.6: Cada uma dessas versões trouxe atualizações contínuas voltadas à segurança, compatibilidade com SQL e desempenho sob cargas maiores, consolidando a evolução do banco em termos de confiabilidade e recursos avançados (MARIADB, 2025).
- MariaDB 10.11: Atualmente, essa é a versão estável mais recente do MariaDB, que reúne todas as melhorias acumuladas nas versões anteriores, oferecendo uma plataforma robusta, segura e altamente eficiente para o gerenciamento de dados (MARIADB, 2025).

1.18 – CANVA

O Canva é uma plataforma *online* de *design* gráfico que permite criar diversos tipos de projetos visuais, como *posts* para redes sociais, apresentações, entre outros. Além disso, o Canva é conhecido por ser intuitivo e fácil de usar (MOBILLS, 2023).

O Canva foi fundado em 2013 por Melanie Perkins e Cliff Obrecht, com o objetivo de simplificar e democratizar o *design* gráfico. Melanie teve a ideia quando percebeu a dificuldade dos seus alunos em usar ferramentas de *design* complexas e quis criar uma plataforma acessível para todos. Antes do Canva, ela e Cliff lançaram o "Fusion Books", uma ferramenta para criar *designs* de anuários escolares, que serviu como base para o Canva. Desde o lançamento, o Canva cresceu rapidamente no mundo todo, e hoje é uma das plataformas de *design* mais populares. Evolução do Canva (ONOVOMERCADO, 2025):

- 2013 – Lançamento do Canva: O Canva foi lançado oficialmente para uma lista de espera. Desde o início, a plataforma tinha como objetivo tornar o *design* algo mais acessível (CANVA, 2023).
- 2014 – Expansão e Novos Recursos: Foi lançado o *marketplace* de *design*, que permitia a contribuição de profissionais, dos quais ganhavam *royalties* sempre que seus *designs* fossem comprados. Além disso, o Canva lançou seu aplicativo para iPad (CANVA, 2023).
- 2015 – Crescimento Rápido: Ao atingir 1 milhão de usuários, a plataforma continuou a adicionar novos recursos e *templates* para atender às necessidades crescentes de sua base de usuários (CANVA, 2023).
- 2016 – Lançamento para Iphone: O lançamento do app do Canva para Iphone deixou-o mais acessível e aumentou sua base de usuários (CANVA, 2023).
- 2017 – Canva Print, lançamento para Android: O Canva Print introduziu de impressão aos *designs* realizados na ferramenta. O Canva também foi lançado como app para *android*, além de ter novas funcionalidades como as animações, além de tornar o app mais acessível, lançando-o em mais de 100 idiomas. (CANVA, 2023)
- 2018 - Lançamento das apresentações, a aquisição do Zeetings como a primeira e o lançamento do Canva na China (CANVA, 2023).
- 2019 - Aquisição do Pexels e do Pixabay, lançamento dos apps no Canva e do Canva para Educação, levando o poder da comunicação visual para a sala de aula (CANVA, 2023).
- 2020 - Expansão Global: Atingindo 30 milhões de usuários mensais, o Canva lançou novos recursos, como vídeos e apresentações interativas além de inaugurar a sua primeira sede nos EUA (CANVA, 2023).
- 2021 - Aquisição do Mockups e da Kaleido e reformulação da marca, com uma nova logo (CANVA, 2023).
- 2022 - Lançamento do Canva para equipes, levando a comunicação visual ao local de trabalho e do Canva Space, um espaço de eventos para a comunidade. Aquisição do Flourish (CANVA, 2023).
- 2023 – 10 Anos do Canva: Com mais de 135 milhões de usuários e 15 bilhões de *designs* criados, o Canva continua a evoluir, adicionando funcionalidades como inteligência artificial para *design* (CANVA, 2023).

1.19 – FIGMA

Figma é uma ferramenta de *design* e prototipagem colaborativa baseada na nuvem, que permite a criação de interfaces de usuário e experiências digitais de forma integrada e em tempo real. Ela reúne funcionalidades de edição vetorial, *design* interativo e colaboração simultânea, sendo utilizada por equipes de *design* em todo o mundo (FIGMA, 2025).

A Figma foi fundada por Dylan Field e Evan Wallace em 2012, com a ideia de deixar o *design* digital mais acessível, possibilitando que qualquer um com acesso à *internet* pudesse participar do processo, independentemente do sistema operacional. A versão beta foi lançada em 2015 e, após testes e ajustes, o produto oficial foi lançado em 2016, ganhando rapidamente popularidade por ser inovador. Evolução do Figma (FIGMA, 2025):

- Setembro de 2016: Lançamento público do Figma, introduzindo uma plataforma de *design* colaborativo baseada em navegador (FIELD, 2024).
- Outubro de 2019: Introdução do Figma *Community*, permitindo que *designers* publiquem e compartilhem seus trabalhos para que outros possam visualizar e adaptar (FIELD, 2024).
- Abril de 2021: Lançamento do FigJam, uma ferramenta de quadro branco digital que facilita a colaboração com notas adesivas, *emojis* e ferramentas de desenho (FIELD, 2024).
- Junho de 2023: Introdução do Dev Mode, projetado para ajudar desenvolvedores a traduzirem *designs* em código de forma mais eficiente (FIELD, 2024).
- Junho de 2024: Lançamento do Figma *Slides*, uma ferramenta de apresentações que permite colaboração em tempo real na criação de *slides* (FIELD, 2024).
- Abril de 2025: Anunciadas atualizações significativas, incluindo melhorias na interface do usuário (UI3), integração de inteligência artificial (Figma *AI*) e aprimoramentos no Dev Mode (FIELD, 2024).

2 – ESTUDO DE CASO

A área de atuação deste projeto está voltada ao desenvolvimento de um sistema *web* que visa modernizar e automatizar as principais funções de uma biblioteca escolar. O objetivo é facilitar o trabalho dos bibliotecários e otimizar a experiência dos usuários por meio de funcionalidades como cadastro de usuários, reservas de exemplares, organização do acervo e controle de empréstimos.

Nos últimos cinco anos, observou-se uma significativa evolução nas bibliotecas, especialmente com a adoção de tecnologias digitais. Plataformas *online* e o uso de *e-books* passaram a ser amplamente utilizados, permitindo empréstimos remotos e leitura digital. Também houve a integração com tecnologias mais avançadas, como sistemas de recomendação baseados em inteligência artificial. A pandemia da COVID-19 acelerou esse processo de transformação digital, ampliando os serviços oferecidos remotamente, como acesso a bancos de dados, revistas e artigos acadêmicos. Além disso, notou-se a implementação de *chatbots*, interfaces mais acessíveis e responsivas, bem como ferramentas de realidade aumentada e realidade virtual, enriquecendo a interação com o acervo.

Entretanto, ainda existem bibliotecas que enfrentam diversos desafios por não contarem com sistemas informatizados. Entre os principais problemas estão a dificuldade na organização do acervo, o controle manual dos prazos de devolução e a limitação na comunicação com os usuários. Além disso, muitas plataformas existentes possuem interfaces obsoletas e pouco intuitivas, o que compromete a eficiência do serviço prestado.

Como solução, este projeto propõe o desenvolvimento de um sistema com *design* moderno e responsivo, que facilite o acesso a todas as funcionalidades em diversos dispositivos. Serão implementados filtros de busca inteligentes, automatização de notificações sobre prazos de devolução por diferentes canais de comunicação (como *e-mail* e SMS), além da criação de um catálogo *online* atualizado e acessível. Essas medidas têm como finalidade tornar a gestão bibliotecária mais ágil, eficiente e adaptada às necessidades atuais.

2.1 – VIABILIDADES

> Estudo de viabilidade é uma análise fundamentada em dados e pesquisas para enxergar os possíveis resultados de uma ação ou projeto, ou seja, observar se é válido ou não determinada ação ou projeto. Esse estudo é extremamente importante para qualquer segmento, pois ajuda a evitar perdas, reduzir gastos, traçar uma estratégia assertiva, enfim, potencializar os ganhos e minimizar os prejuízos (CIMATECJR, 2025).

2.1.1 – Viabilidade operacional

A viabilidade operacional avalia se uma organização possui os recursos, processos e estruturas necessárias para implementar e manter suas atividades com eficiência. Para isso, verifica-se se a infraestrutura disponível — incluindo tecnologia, instalações e ativos materiais — e o quadro de colaboradores estão adequadamente dimensionados e capacitados para atingir as metas propostas. Além disso, analisa-se a forma como as equipes se comunicam e se as rotinas de trabalho estão bem definidas, de modo a evitar gargalos e desperdícios (TACONTRATADO, 2025).

Esse estudo é essencial para qualquer projeto ou negócio, pois evita investimentos em iniciativas inviáveis e direciona esforços para otimizar processos e alocar recursos de maneira mais inteligente. Ao mapear potenciais entraves — como falta de pessoal qualificado, sistemas defasados ou fluxos de trabalho pouco claros — , a organização pode ajustar sua estrutura e seus procedimentos antes de dar início à operação, garantindo maior chance de sucesso e sustentabilidade a longo prazo (TACONTRATADO, 2025).

As tecnologias utilizadas foram: JavaScript, Node.js, HTML, CSS, MYSQL, Hostinger (hospedagem do *site* do banco de dados).

Conhecimentos necessários e dificuldades encontradas:

- Manipulação de marcação HTML e estilização CSS – O grupo possui muito domínio na utilização de HTML, já que há bastante facilidade na criação de páginas. No caso do CSS, o grupo apresenta um conhecimento intermediário, sabendo aplicar estilizações básicas, porém ainda existem dificuldades em criar estilos mais avançados. A estimativa de aprimoramento no CSS é de 1 a 2 semanas.
- Uso de funções no *front-end* com JavaScript – O grupo possui domínio sólido em JavaScript, sabendo aplicar suas funcionalidades de forma eficiente para interação e dinamismo nas páginas.

- Uso de Node.js para desenvolvimento *back-end* – O grupo possui conhecimento introdutório em Node.js, compreendendo seus conceitos básicos como criação de servidores e utilização de pacotes via NPM. No entanto, ainda há dificuldades na integração com *frameworks* como *Express*, na manipulação de rotas e no uso de *middlewares*. A estimativa de aprimoramento é de 1 a 2 semanas com prática e estudo direcionado.
- Banco de Dados MySQL – O grupo tem bom domínio em MySQL, especialmente na criação de banco, tabelas e realização de consultas (*SELECT*, *INSERT*, *UPDATE* e *DELETE*).
- Hospedagem do *site* na Hostinger – O grupo ainda não possui domínio sobre os processos de hospedagem, sendo necessário realizar pesquisas e testes para compreender como funciona o ambiente de hospedagem, configuração de domínio e banco de dados. A estimativa de aprendizado é de aproximadamente 2 semanas.

2.1.2 - Viabilidade técnica

> O estudo de viabilidade técnica analisa os recursos técnicos disponíveis para o projeto. Este estudo determina se temos os equipamentos certos e suficientes e o conhecimento técnico adequado para cumprir os objetivos do projeto (MARTINS, 2025).

Itens físicos:

Computadores com acesso à *internet*, equipados com sistema operacional Windows 7 ou superior, navegador atualizado, *mouse*, teclado e monitor de no mínimo 14 polegadas. Esses equipamentos são fundamentais para o desenvolvimento do sistema, tanto na parte de programação quanto na administração do banco de dados e testes. Caso ocorram problemas técnicos nas máquinas, serão contratados serviços de assistência técnica para solucionar rapidamente.

Ferramentas digitais:

- Visual Studio Code (*VS Code*) – Editor de código gratuito e *open source*. Essencial para o desenvolvimento do projeto, pois possui recursos como controle de versionamento Git integrado, realce de sintaxe, complementação inteligente, depuração e diversos *plugins* que facilitam o desenvolvimento em HTML, CSS e JavaScript (CODEVISUALSTUDIO, 2025).

- XAMPP – Ferramenta gratuita e *open source* que simula um servidor local, permitindo executar e testar aplicações *web* com suporte a PHP e MySQL. É essencial para os testes antes da hospedagem do *site* (APACHEFRIENDS, 2024).
- Node.js – Ambiente de execução JavaScript *server-side*, utilizado no desenvolvimento do *back-end* da aplicação. Permite a criação de servidores, manipulação de APIs e integração com bancos de dados de forma eficiente e escalável. Sua estrutura modular baseada em pacotes (via npm) facilita a organização e o reuso de código (NODEJS, 2025).
- Figma – Plataforma gratuita (na versão limitada, suficiente para o projeto) utilizada na criação de protótipos de telas e interfaces, facilitando o planejamento visual e a experiência do usuário (FIGMA, 2025).
- Canva – Ferramenta gratuita que será utilizada na criação de apresentações, *banners*, imagens e elementos gráficos necessários para documentações, apresentações e divulgação do projeto (CANVA, 2025).
- Hostinger – Serviço de hospedagem *web* utilizado para tornar o *site* acessível na *internet*. A equipe ainda precisa aprofundar o conhecimento sobre como funciona o processo de hospedagem, configurações de domínio, certificados SSL (*Secure Sockets Layer* – Camada de Soquete Seguro) e conexão do banco de dados com o servidor (HOSTINGER, 2025).
- MySQL – Sistema de gerenciamento de banco de dados relacional, fundamental para armazenar e gerenciar as informações do sistema, como cadastro de livros, usuários, empréstimos, avaliações e comentários (ERICSON, 2024).

Medidas de segurança e soluções:

- Segurança dos dados – A hospedagem na Hostinger conta com certificado TLS (*Transport Layer Security* – Segurança de Camada de Transporte) /SSL, que oferece criptografia nas comunicações, protegendo dados sensíveis dos usuários, como informações pessoais e de empréstimos (HOSTINGER, 2025).
- Queda da hospedagem (Hostinger) – Em caso de instabilidade na hospedagem, será acionado o suporte técnico da Hostinger, que oferece atendimento 24h. Além disso, serão realizados *backups* diários do banco de dados e dos arquivos do *site*, para garantir que não haja perda de dados. Caso ocorram problemas

recorrentes, há possibilidade de migração para outros serviços de hospedagem (RECLAMEAQUI, 2025).

- Falhas no banco de dados – Serão realizados *backups* automáticos frequentes, além de otimização das consultas SQL e uso de índices para manter o desempenho e evitar problemas de lentidão ou falhas.
- Manutenção e atualizações – O projeto contará com um sistema de versionamento de código utilizando Git e GitHub, permitindo que qualquer alteração seja controlada, revertida e documentada. Haverá também uma documentação técnica completa, descrevendo configurações, dependências e procedimentos, para facilitar atualizações futuras e manutenções corretivas ou evolutivas.

2.1.3 - Viabilidade de cronograma

> A viabilidade de cronograma avalia a probabilidade de um projeto ser concluído dentro do prazo proposto. Esse é um estudo de viabilidade crucial, pois os resultados determinarão se o projeto terá sucesso. Ele identifica os principais entraves da iniciativa que podem afetar o cronograma, incluindo restrições internas e externas, como normas, políticas, orçamentos e tecnologia (LUCID, 2025).

Tabela 1 - Cronograma

Responsável	Tarefas	Datas Inicio	Datas Fim	Fevereiro	Março	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro	Novembro
Joyce	Pesquisa e Justicativa do tema	07/fev	14/fev										
Todos	Revisão Sistematica	07/fev	21/fev										
Camila e Juan	Definição da Paleta de Cores	07/fev	21/fev										
Ana Laura	pesquisa de viabilidade (operacional)	21/fev	26/fev										
Guilherme	pesquisa de viabilidade (economia)	21/fev	27/fev										
Vinicius Ramalho	pesquisa de viabilidade (tecnica)	21/fev	27/fev										
Camila e Ana Laura	Levantamento de Requisitos (entrevista)	28/fev	07/mar										
Juan,Venicius	Levantamento de Requisitos (questionario)	28/fev	07/mar										
Guilherme e Vinicius	Levantamento de Requisitos (leis)	28/fev	07/fev										
Guilherme e Vinicius	Levantamento de Requisitos analise do texto)	07/mar	12/mar										
Camila,Juan,Ramalho	persona	21/mar	28/mar										
Camila,Juan	Criação do Prototipo(base)	22/fev	07/mar										
Camila,Juan	Criação do Prototipo(melhoria)	07/mar	21/mar										
Vinicius R., Venicius	Definição da Linguagem de Progrmação	07/fev	21/fev										
Joyce	Estrutura do Banco de Dados	01/mar	21/mar										
Vinicius R.	Desenvolvimento do Html e Css	11/abr	12/mai										
Venicius	Desenvolvimento do Login	04/mai	07/jun										
Ana e Guilherme	Capitulo 1	21/mar	11/abr										
Venicius	Desenvolvimento do Cadastro de Livros	30/mai	18/abr										
Camila e Juan	Escolha da Identidade Visual (fontes, ícones)	20/abr	25/abr										
Camila e Juan	Padronização com as Heurísticas de Nielsen	23/mai	04/jun										
Joyce	Testes e Ajustes de Desempenho do Banco	26/abr	30/abr										
Vinicius R.	Implementação do Cadastro de Usuários	01/mai	30/abr										
Camila e Juan	Ajustes Visuais da Aplicação	20/mai	30/mai										
Ana e Guilherme	Capitulo 2	11/abr	27/jun										
Venicius	Implementação do Sistema de Empréstimos	01/jun	19/jun										
Joyce	Backup e Segurança de Dados	20/jun	04/jul										
Camila e Juan	Refinamento do Design	15/jul	28/jul										
Ana e Guilherme	Escrita do Relatório Final	29/ago	29/ago										
Ana e Guilherme	Capitulo 3	26/set	26/set										
Todos	Testes e Melhorias no Sistema	03/out	03/out										
Todos	Ultimos Ajustes	10/out	10/out										
Todos	Preparação Para Apresentação	07/nov	07/nov										
Todos	Apresentação	21/nov	21/nov										

FONTE: AUTORIA PRÓPRIA (2025)

A Tabela 1 apresenta o cronograma detalhado de execução do projeto de desenvolvimento de sistema, descrevendo de forma estruturada todas as etapas previstas para sua realização. Como pode ser observado na imagem capturada em 06 de junho de 2025, no presente momento, diversas atividades já se encontram

concluídas (em verde), enquanto outras permanecem em andamento (em amarelo), ou em atraso (em vermelho), conforme a complexidade inerente de cada fase. Esse panorama visual permite acompanhar o progresso do projeto em tempo real, facilitando a identificação de gargalos e a tomada de decisões estratégicas.

O planejamento iniciou-se com as atividades preliminares de definição e fundamentação do tema, incluindo a pesquisa e justificativa da escolha, revisão sistemática da literatura e estudos de viabilidade operacional, técnica e econômica. Esta etapa teve como objetivo estabelecer as bases teóricas e práticas essenciais para o desenvolvimento do sistema proposto, permitindo uma compreensão ampla do problema e das possíveis soluções tecnológicas aplicáveis. A consistência desse embasamento é crucial para orientar as próximas fases com clareza e segurança.

Em sequência, foi realizada a fase de levantamento de requisitos, abordando aspectos técnicos, legais, operacionais e de usabilidade, essenciais para delinear as funcionalidades, restrições e características específicas do sistema. Com os requisitos definidos, procedeu-se à elaboração dos protótipos e desenvolvimento da arquitetura do sistema, contemplando tanto a modelagem do banco de dados quanto a estruturação das linguagens de programação e da interface do usuário. Essa fase serviu também como ponto de validação com os usuários, possibilitando ajustes antecipados que reduziram riscos de retrabalho nas fases seguintes.

O cronograma contempla ainda a execução de atividades de implementação e desenvolvimento funcional, tais como: cadastro de livros e usuários, sistema de empréstimos, *login*, ajustes visuais e padronização segundo as heurísticas de Nielsen, com o objetivo de garantir a eficiência, segurança e usabilidade da aplicação desenvolvida. Além disso, o foco em boas práticas de codificação e organização do projeto colabora para sua manutenção futura e escalabilidade, caso desejado.

Em paralelo ao desenvolvimento técnico, são executadas as fases de testes, ajustes de desempenho e refinamento das funcionalidades, assegurando a estabilidade e o correto funcionamento do sistema. Nas etapas subsequentes, encontram-se previstas a redação e revisão dos capítulos do Trabalho de Conclusão de Curso (TCC), bem como a preparação da apresentação final, respeitando os prazos estabelecidos no planejamento inicial. Essa preparação envolve também a

organização dos materiais visuais e a simulação da defesa, de modo a garantir clareza na comunicação dos resultados alcançados.

Cabe destacar que o cronograma foi elaborado com base em estimativas realistas de tempo e recursos, considerando a complexidade de cada tarefa e a disponibilidade dos membros da equipe. O planejamento permanece sujeito a ajustes contínuos, a fim de garantir a adequada execução e conclusão do projeto dentro do prazo estabelecido. A flexibilidade desse planejamento tem se mostrado essencial para lidar com imprevistos e otimizar o desempenho coletivo ao longo da jornada acadêmica.

2.1.4 - Viabilidade de econômica

> A viabilidade econômica é a capacidade de um projeto, empreendimento ou investimento gerar lucro e retornos financeiros positivos. Em outras palavras, é a análise de verificação se uma ação é economicamente justificável e se vale a pena do ponto de vista financeiro. A viabilidade econômica envolve a avaliação de diversos aspectos financeiros, como custos iniciais, receitas esperadas, despesas operacionais, retorno sobre o investimento e período de recuperação do investimento. Além disso, fatores externos, como o ambiente econômico, a concorrência e as tendências do mercado, também são considerados na análise de viabilidade econômica. Um projeto é considerado viável economicamente quando seus benefícios financeiros superam os custos envolvidos e geram um retorno satisfatório para os investidores (LEMMI, 2024).

Tabela 2 - Viabilidade Econômica

Viablididade Econômica

CUSTOS	MENSAL	ANUAL
Aluguel	R$ 2.000,00	R$ 24.000,00
Internet	R$ 300,00	R$ 3.600,00
Salários	R$ 20.170,22	R$ 242.042,64
Hospedagem	R$ 10,99	R$ 131,88
Água	R$ 200,00	R$ 2.400,00
Energia	R$ 900,00	R$ 10.800,00
Manutenção e limpeza	R$ 1.000,00	R$ 12.000,00
Domínio	R$ 3,33	R$ 40,00
	MENSAL	ANUAL
CUSTO TOTAL:	R$ 29.876,21	R$ 358.514,52

CUSTOS ÚNICOS	
Hardware	R$ 28.000,00
Mobília	R$ 1.000,00
Divulgação	R$ 34.500,00

BIBLIOTECAS	ALUGUEL MENSAL	TOTAL MENSAL
184	R$ 200,00	R$ 36.800,00

LUCROS	MENSAL	ANUAL
Receita total	R$ 36.800,00	R$ 441.600,00
Custo total	R$ 29.876,21	R$ 358.514,52
Lucro líquido	R$ 6.923,79	R$ 83.085,48

FONTE: AUTORIA PRÓPRIA (2025)

A Tabela 2 apresenta um levantamento detalhado da viabilidade econômica do projeto, contemplando os custos mensais, anuais, os investimentos únicos e a previsão de receita. Entre os principais custos mensais estão despesas como aluguel, *internet*, salários da equipe, energia, água, limpeza e manutenção de equipamentos. Os valores classificados como "custos únicos", como aquisição de *hardware*, mobília

e ações de divulgação, foram divididos em 12 parcelas e incorporados aos custos mensais totais, além de terem sido somados de forma integral ao cálculo dos custos anuais, permitindo uma análise mais realista do impacto financeiro e totalizando aproximadamente R$ 29.876,21 por mês, o que representa um custo anual de R$ 358.514,52.

O modelo de negócio prevê a implantação do sistema em 184 bibliotecas, com uma mensalidade de R$ 200,00 por unidade, o que resultará em uma receita bruta mensal de R$ 36.600,00 e uma receita anual de R$ 441.600,00. Dessa forma, considerando o total de custos anuais de R$ 358.514,52, o lucro anual estimado do projeto será de aproximadamente R$ 83.085,48, representando a diferença entre a receita bruta anual e os custos totais.

Além disso, está prevista uma campanha de divulgação no valor de R$ 34.500,00, com o objetivo de promover o sistema e ampliar sua adesão. A estratégia de implantação inicial foi pensada para atender as bibliotecas de maneira personalizada, com possibilidade de futuras expansões para outras instituições educacionais.

2.2 – TÉCNICA DE LEVANTAMENTO DE REQUISITOS

> As técnicas de levantamento de requisitos têm por objetivo superar as dificuldades relativas a esta fase. Todas as técnicas possuem um conceito próprio e suas respectivas vantagens e desvantagens, que podem ser utilizadas em conjunto pelo analista (DEVMEDIA, 2025).

2.2.1 – Entrevista

Foi realizada uma entrevista presencial com alunos, quatro funcionários da ETEC Euro Albino de Souza e quatro bibliotecários. As entrevistas foram compostas por perguntas direcionadas conforme o perfil de cada grupo, sendo elaboradas com o objetivo de entender as necessidades, dificuldades e sugestões relacionadas ao uso do sistema da biblioteca.

As perguntas foram feitas com a intenção de coletar dados reais sobre o funcionamento atual da biblioteca, avaliar o interesse do público por um sistema digital mais interativo e identificar funcionalidades que poderiam facilitar o acesso e o controle de empréstimos de livros. Essa coleta de informações foi fundamental para

embasar o desenvolvimento do projeto, tornando-o mais próximo da realidade da instituição e das pessoas que o utilizarão.

2.2.1.1 – Entrevista com os alunos

A entrevista com os alunos foi realizada presencialmente, abrangendo oito perguntas específicas, cada uma elaborada com um objetivo distinto. Essa abordagem permitiu uma interação mais próxima e personalizada, possibilitando aos entrevistadores obter informações valiosas sobre as experiências, preferências e dificuldades enfrentadas pelos alunos no uso da biblioteca. No total foram cinco alunos.

Perguntas:

1 - Como você costuma procurar livros na biblioteca?

Justificativa: Compreender os critérios de pesquisa mais utilizados pelos usuários (gênero, autor, título, etc.) permite otimizar os filtros do sistema da biblioteca. Isso ajudará a tornar a pesquisa mais intuitiva e eficiente, facilitando a localização dos livros desejados.

2 – Quais dificuldades você já enfrentou ao tentar encontrar ou pegar um livro emprestado?

Justificativa: identificar os desafios enfrentados pelos usuários ao buscar e pegar livros emprestados possibilita melhorias na organização do acervo e no funcionamento do sistema de empréstimo, tornando a experiência mais fluida e acessível.

3 – O que influencia na escolha do livro?

Justificativa: Saber quais fatores influenciam a escolha dos livros (indicações, sinopse, capa, gênero favorito) permite aprimorar a experiência do usuário, indicando livros de formato mais personalizados e eficientes no sistema da biblioteca.

4 – Quais dificuldades você enfrenta ao procurar livros na biblioteca?

Justificativa: Compreender os principais obstáculos enfrentados na busca por livros, como a disposição dos exemplares ou a falta de organização das prateleiras, pode orientar melhorias estruturais na biblioteca e otimizações no catálogo digital.

5 - Gostaria de ter acesso digital à biblioteca (*site*/app para reservas)?

Justificativa: Avaliar o interesse dos usuários em uma plataforma digital auxilia no planejamento e desenvolvimento de um sistema que permite maior autonomia na consulta ao acervo, facilitando reservas e reduzindo o tempo gasto na escolha dos livros presencialmente.

6 – O que acha da ideia de uma seção de comentários sobre os livros?

Justificativa: Compreender a percepção dos usuários sobre a inclusão de comentários pode ajudar na implementação de um sistema que facilita a troca de opiniões e experiências sobre os livros, incentivando o engajamento e auxiliando na tomada de decisão.

7 - Se a biblioteca te indicasse livros parecidos com os que você já pegou antes, você ia curtir?

Justificativa: Essa pergunta busca entender se os alunos se interessariam por um sistema de recomendações personalizadas, com base no histórico de empréstimos. A ideia é tornar a experiência na biblioteca mais atrativa e interativa, ajudando os usuários a descobrirem novos títulos que combinem com seus gostos, assim como já acontece em plataformas de *streaming* e leitura digital.

8 – Quais funcionalidades gostariam que fossem adicionadas?

Justificativa: Permitir que os próprios usuários sugiram funcionalidades que ajudem a desenvolver um sistema que atenda melhor às suas necessidades, garantindo que uma biblioteca digital seja útil e acessível para diferentes perfis de leitores.

2.2.1.2 – Entrevistas com professores

A entrevista presencial realizada com três professoras da ETEC Euro Albino de Souza contou com sete perguntas cuidadosamente elaboradas, cada uma com um objetivo específico. Essa abordagem permitiu que os entrevistadores obtivessem uma visão mais completa e detalhada das necessidades, experiências e perspectivas das docentes, tornando-se um instrumento eficaz para entender como a biblioteca pode ser mais útil no ambiente escolar.

Perguntas:

Como você utiliza os recursos da biblioteca nas suas aulas?

Justificativa: Compreender como os professores ocupam a biblioteca permite identificar oportunidades para ampliar seu papel no apoio pedagógico. Isso ajuda a alinhar o acervo e as funcionalidades do sistema às necessidades dos educadores e alunos.

2 - Quais dificuldades você encontra para recomendar leituras para seus alunos?

Justificativa: identificar os desafios enfrentados pelos professores na recomendação de livros possibilita a criação de estratégias para incentivo à leitura, seja por meio de melhorias na organização do acervo, divulgação de títulos ou uso de ferramentas digitais.

3 – O que poderia facilitar o processo de indicação e reserva de livros para os alunos?

Justificativa: Conhecer as necessidades dos professores no processo de indicação e reserva auxilia no desenvolvimento de recursos que tornam esse processo mais prático e acessível, aumentando a efetividade das recomendações.

4 – Você gostaria de um sistema que sugerisse materiais didáticos baseados no conteúdo que você ensina?

Justificativa: Implementar um sistema de recomendações personalizadas pode melhorar o uso da biblioteca como ferramenta pedagógica, oferecendo materiais atualizados ao currículo e às preferências dos alunos.

5 – Você gostaria que tivesse uma funcionalidade para criar listas de leitura para os alunos dentro do sistema da biblioteca, você usaria?

Justificativa: A possibilidade de criar listas de leitura facilita a organização de conteúdos sugeridos ou obrigatórios, permitindo que os alunos acessem facilmente o horário e planejem melhor seus estudos.

6 – Você acredita que uma biblioteca poderia ter um papel mais ativo no apoio ao ensino? Se sim, como?

Justificativa: Explorar formas de ampliar o impacto da biblioteca no ensino possibilita transformá-la em um espaço mais sonoro e atraente para os alunos, incentivando hábitos de leitura e pesquisa.

7- O que você acha da ideia de uma aba exclusiva para professores indicarem livros e acompanharem os alunos?

Justificativa: Criar um espaço dedicado aos professores no sistema da biblioteca facilita a curaria de conteúdos e o acompanhamento do engajamento dos alunos, tornando o processo de ensino mais interativo e personalizado.

2.2.1.3 – Entrevista com o diretor

A entrevista com o diretor da ETEC Euro Albino de Souza foi realizada de forma presencial, com o objetivo de compreender a visão da gestão sobre a biblioteca escolar e suas possibilidades de modernização. O diretor foi escolhido para essa entrevista por ser o principal responsável pelas decisões administrativas da instituição, além de ser a pessoa que pode autorizar, apoiar e buscar recursos para a implementação de melhorias no sistema bibliotecário. Sua participação foi essencial para entender os desafios institucionais, a viabilidade de propostas tecnológicas e o papel da biblioteca no processo educacional sob a perspectiva da liderança escolar.

Perguntas:

1 – Qual a sua visão sobre o papel da biblioteca no processo educacional da escola?

Justificativa: Essa pergunta visa compreender como a direção enxerga a importância da biblioteca no apoio ao ensino-aprendizagem. Serve como base para avaliar o potencial que a gestão reconhece na biblioteca como ferramenta complementar ao conteúdo trabalhado em sala de aula.

2 – Você considera importante a criação de um sistema digital que integre alunos, professores e a biblioteca? Por quê?

Justificativa: Essa pergunta busca avaliar a receptividade da direção em relação ao uso da tecnologia para aproximar a comunidade escolar da biblioteca, promovendo um ambiente mais conectado, acessível e eficiente.

3 – Que tipo de resultado ou indicadores você gostaria de acompanhar sobre o uso da biblioteca?

Justificativa: Tem como objetivo entender quais métricas a direção considera relevantes para tomar decisões e planejar ações futuras, como aquisição de acervo, frequência dos usuários e livros mais acessados.

4 – Quais seriam os principais desafios administrativos para modernizar o sistema da biblioteca?

Justificativa: Essa pergunta permite identificar possíveis barreiras à implantação do sistema digital, como questões de pessoal, orçamento ou infraestrutura, ajudando a propor soluções realistas e viáveis.

5 – Você acredita que a modernização da biblioteca pode impactar positivamente o desempenho dos alunos? De que forma?

Justificativa: O intuito é compreender se a direção acredita que a transformação digital pode contribuir com o desenvolvimento acadêmico dos alunos, promovendo acesso facilitado à informação e recursos complementares.

6 – Como a direção poderia apoiar a capacitação dos professores e funcionários no uso de um novo sistema de biblioteca?

Justificativa: A pergunta busca explorar formas de envolver a gestão na capacitação da equipe, garantindo que todos saibam utilizar o sistema de forma eficiente e autônoma, sem sobrecarregar os profissionais.

7 – A direção estaria disposta a investir ou buscar parcerias para o desenvolvimento desse sistema digital?

Justificativa: Tem como objetivo verificar a abertura da gestão para investir recursos ou buscar apoio externo (como parcerias ou patrocínios), algo essencial para viabilizar o desenvolvimento e a implementação do projeto.

8 – O que você gostaria que tivesse no sistema para adquirir?

Justificativa: Essa pergunta busca levantar sugestões da gestão sobre as funcionalidades que poderiam ser incluídas no sistema, como acervo digital, TCCs, artigos e produções dos alunos, ampliando o valor da plataforma para toda a escola.

2.2.1.4 – Entrevistas com os bibliotecários

A entrevista com os bibliotecários de quatro bibliotecas escolares (Etec Mogi Guaçu, Etec Mogi Mirim, SENAC e SENAI) foi fundamental para entender como funcionam, na prática, os sistemas de gerenciamento utilizados atualmente. Através do contato direto com os profissionais que lidam diariamente com empréstimos, devoluções, catalogação e organização do acervo, foi possível identificar falhas, limitações, sugestões e necessidades específicas que não são percebidas apenas por observação externa. Essas informações ajudam a embasar o desenvolvimento de um novo sistema mais moderno, acessível e eficiente, que atenda tanto às demandas técnicas dos bibliotecários quanto à praticidade para os alunos. Além disso, permite alinhar o projeto com as rotinas reais das bibliotecas escolares, tornando-o viável e útil para a comunidade escolar.

1 - Como funciona o sistema da sua biblioteca?

Justificativa: Entender como o sistema atual opera nos permite identificar seus pontos fortes e fracos. Com isso, pode-se estruturar um sistema eficiente, aproveitando boas práticas e corrigindo falhas que dificultam o dia a dia dos bibliotecários e alunos.

2 – Quais são os recursos que faltam ou que você gostaria que tivesse?

Justificativa: Identificar as funcionalidades ausentes ou insuficientes no sistema atual ajuda a projetar um sistema mais completo e útil. Isso garante que o novo sistema atenda às necessidades reais dos usuários, trazendo melhorias que façam diferença no cotidiano da biblioteca.

3 – Qual é a maior dificuldade que você enfrenta com esse sistema?

Justificativa: Saber quais desafios os bibliotecários enfrentam permite criar soluções práticas e eficientes. Dessa forma, pode-se eliminar processos complicados, melhorar a usabilidade do sistema e garantir um fluxo de trabalho mais ágil e produtivo.

4 – Você já teve problemas de usabilidade ao utilizar o sistema? Se sim, quais?

Justificativa: A usabilidade impacta diretamente a eficiência e a experiência dos usuários. Identificar dificuldades no uso do sistema ajuda a desenvolver uma interface mais acessível e intuitiva, reduzindo erros e facilitando a navegação para bibliotecários e alunos.

5 – Quais funcionalidades do sistema atual você mais utiliza?

Justificativa: Compreender as funções mais utilizadas permite focar esforços no aprimoramento dessas áreas, tornando-as mais intuitivas e eficientes. Além disso, pode-se eliminar funcionalidades desnecessárias, simplificando o sistema sem comprometer seu desempenho.

6 – Como você avalia o desempenho do sistema em termos de velocidade e confiabilidade?

Justificativa: Um sistema lento ou instável pode prejudicar o funcionamento da biblioteca, causando atrasos e frustrações. Avaliar o desempenho do sistema atual nos ajuda a garantir que o novo sistema seja ágil, confiável e capaz de atender à demanda da biblioteca sem falhas.

7 – Á alguma funcionalidade do sistema atual que você acha desnecessária ou que poderia ser aprimorada?

Justificativa: Identificar funções pouco utilizadas ou que precisam de melhorias permite otimizar o sistema, tornando-o mais eficiente e livre de recursos que ocupam espaço sem agregar valor. Isso resulta em uma experiência mais fluida para todos os usuários.

8 – Como é o processo de gerenciamento de livros, incluindo aquisição, catalogação e empréstimo? O sistema facilita esses processos?

Justificativa: Compreender como a biblioteca gerencia seus livros nos ajuda a projetar um sistema que simplifique esses processos. Um sistema bem estruturado deve facilitar a catalogação, otimizar a busca por livros e automatizar tarefas burocráticas, tornando a gestão mais eficiente.

2.2.2 – Questionário

Foram realizadas 8 perguntas objetivas no Google Docs. e compartilhada aos alunos da escola ao qual 39 deles responderam. Abaixo há uma análise/objetivo de cada pergunta e de cada alternativa disponibilizada.

Perguntas:

1 - Você usaria um *site* para a biblioteca?

Opção 1: Sim.

Opção 2: Não.

Essa pergunta avalia o interesse dos usuários por um sistema *online*, ajudando a entender se há demanda real por uma plataforma digital de apoio à biblioteca.

2 - Você gostaria de reservar livros diretamente pelo *site*?

Opção 1: Sim.

Opção 2: Não.

A reserva *online* é uma funcionalidade comum em bibliotecas modernas. Saber se o público considera isso útil ajuda a definir se ela deve ser prioridade no desenvolvimento do sistema.

3 - Você acha que seria útil reservar um livro *online* antes de ir à biblioteca?

Opção 1: Sim.

Opção 2: Não.

Essa pergunta complementa a anterior, focando na praticidade e na organização do tempo dos usuários, além de verificar se essa função agregaria valor ao sistema.

4 - Você acha útil poder ler avaliações de outros leitores sobre os livros em um catálogo *online*?

Opção 1: Sim.

Opção 2: Não.

Essa funcionalidade incentiva a troca de experiências entre os leitores e pode ajudar na escolha de novos livros, promovendo mais engajamento com o acervo.

5 - Você tem dificuldades de encontrar livros específicos que deseja na biblioteca?

Opção 1: Sim.

Opção 2: Não.

Identificar esse tipo de dificuldade ajuda a entender a necessidade de um sistema com busca eficiente e categorização clara, facilitando a localização de obras.

6 - Você acha importante consultar a disponibilidade de livros no catálogo *online*?

Opção 1: Sim.

Opção 2: Não.

Saber se o livro está disponível evita deslocamentos desnecessários e melhora a organização da biblioteca. Essa pergunta verifica a utilidade prática dessa função para os usuários.

7 - Você utilizaria um sistema de recomendações de livros baseado no seu histórico de buscas?

Opção 1: Sim.

Opção 2: Não.

A recomendação personalizada pode incentivar a leitura e aumentar o uso do sistema. A pergunta verifica o interesse dos usuários por esse tipo de tecnologia.

8 - Você já utilizou algum sistema *online* para buscar livros?

Opção 1: Sim.

Opção 2: Não.

Essa pergunta investiga o nível de familiaridade do público com ferramentas digitais de biblioteca, o que ajuda a projetar um sistema com uma interface adequada e acessível.

2.2.3 - Análise de texto

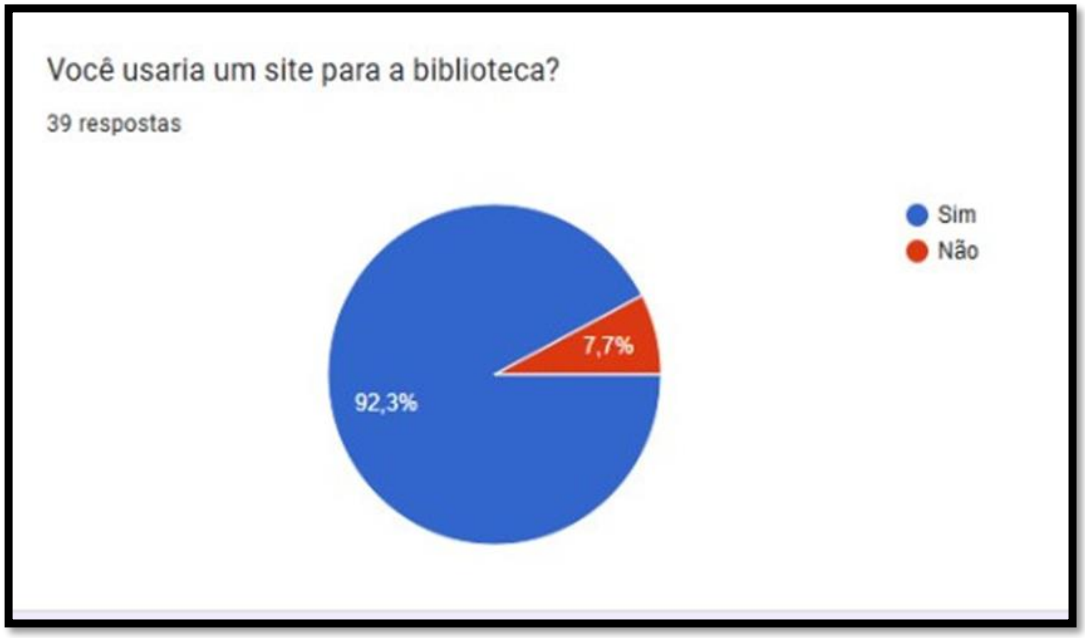
A análise de texto apresenta as principais conclusões e recomendações obtidas a partir de um questionário *online* aplicado a alunos da escola. O objetivo foi entender as necessidades e preferências dos usuários em relação ao uso de um sistema digital para biblioteca. As respostas coletadas ajudaram a identificar aspectos importantes para o direcionamento do desenvolvimento do projeto.

2.2.3.1 - Análise das respostas do questionário

O questionário, com 8 perguntas objetivas, foi respondido por 39 alunos, abordando o interesse e as dificuldades no uso da biblioteca atual. A maioria demonstrou apoio à criação de um *site*, com funcionalidades como reserva *online*, consulta de disponibilidade e avaliações de livros. As respostas mostraram diferentes níveis de familiaridade com tecnologia, reforçando a importância de uma plataforma intuitiva e acessível. Esses dados servirão de base para decisões sobre as funcionalidades e a usabilidade do sistema, como demonstrado nos gráficos com os percentuais de cada resposta.

Resposta pergunta 1:

Figura 1 - Pergunta 1 Questionário



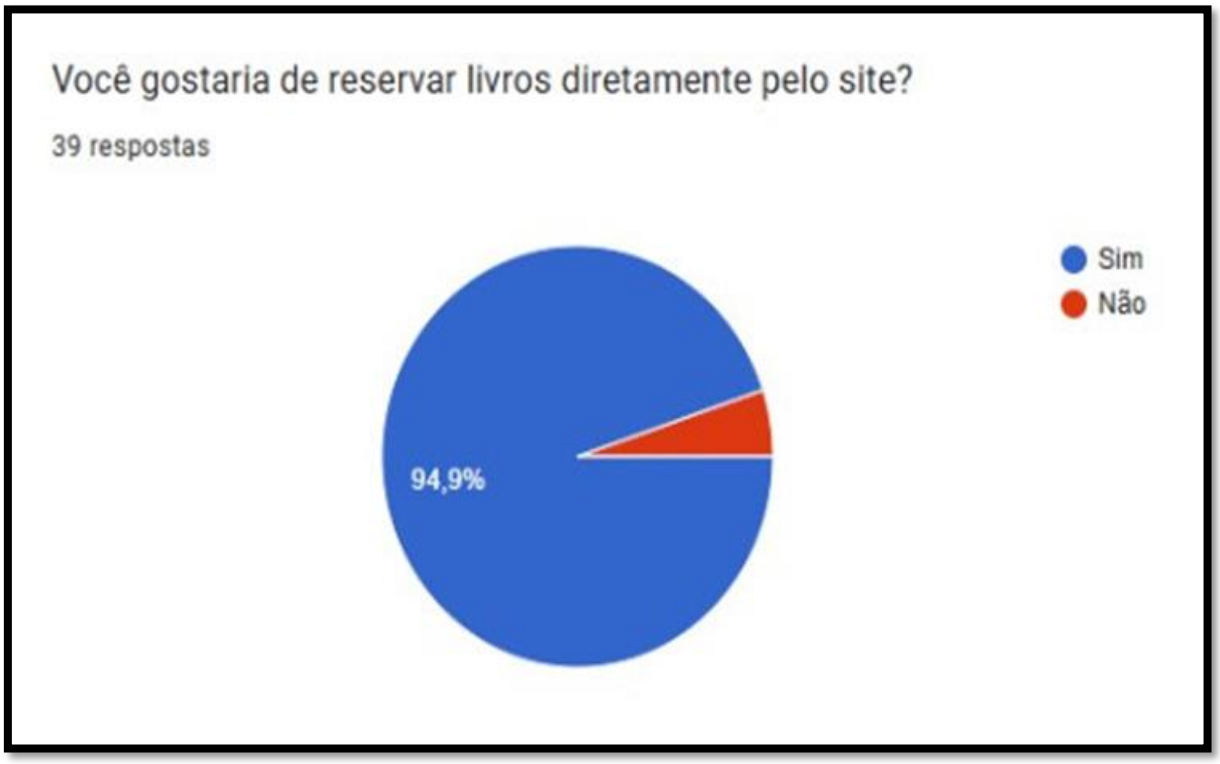
FONTE: AUTORIA PRÓPRIA (2025)

O gráfico apresentado na Figura 1 mostra a porcentagem de alunos que afirmaram se utilizariam um *site* da biblioteca escolar.

Resposta – A maioria dos alunos (92,3%) respondeu “Sim”, o que demonstra uma alta aceitação à criação de uma plataforma digital para facilitar o acesso às funcionalidades da biblioteca. Apenas 7,7% responderam “Não”, mostrando que o projeto tem grande potencial de adesão.

Resposta pergunta 2:

Figura 2 - Pergunta 2 Questionário



FONTE: AUTORIA PRÓPRIA (2025)

A Figura 2 representa as respostas sobre o interesse dos alunos em realizar reservas de livros por meio de um *site*.

Resposta – Com 94,9% das respostas indicando “Sim”, percebe-se uma forte demanda por essa funcionalidade. Isso destaca a importância de incluir o recurso de reserva *online* no sistema proposto. Apenas 5,1% responderam “Não”, destacando que a maioria dos usuários gostariam de utilizar uma funcionalidade desse tipo, sendo ela então importante para o sistema e uma funcionalidade, uma vez que é requisitada

por muitas das pessoas que fizeram a pesquisa. Portanto, é necessária a inclusão dela.

Resposta pergunta 3:

Figura 3 - Pergunta 3 Questionário



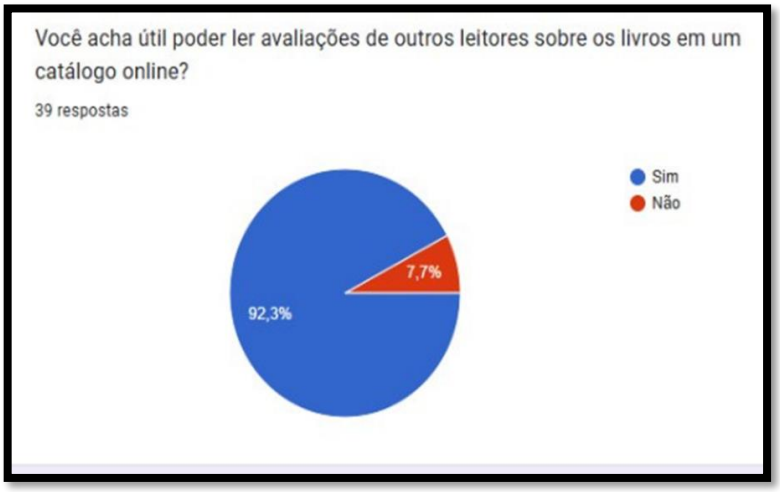
FONTE: AUTORIA PRÓPRIA (2025)

O gráfico da Figura 3 mostra a opinião dos alunos sobre a utilidade de reservar livros remotamente antes de comparecer à biblioteca.

Resposta – A aprovação dessa funcionalidade foi quase unânime: 97,4% dos alunos consideram útil a possibilidade de reservar livros *online*, o que reforça a relevância dessa função no projeto.

Resposta pergunta 4:

Figura 4 - Pergunta 4 Questionário



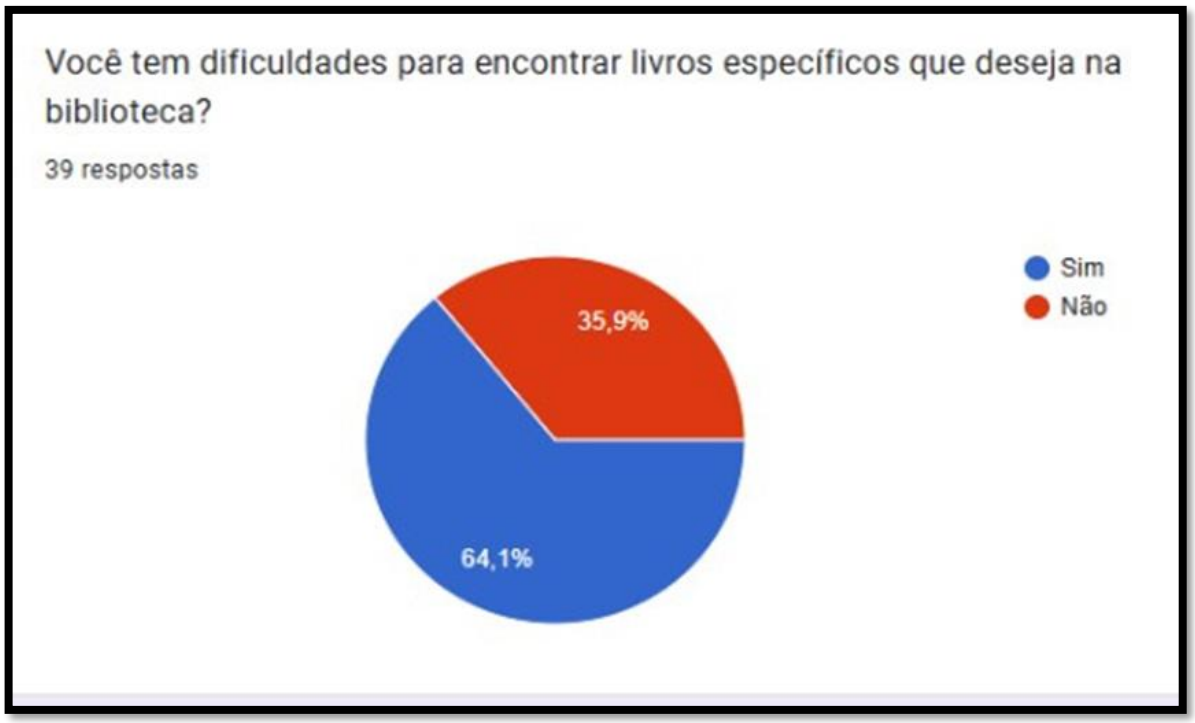
FONTE: AUTORIA PRÓPRIA (2025)

A Figura 4 mostra a percepção dos alunos sobre a presença de avaliações de outros leitores no sistema da biblioteca.

Resposta – Com 92,3% de aprovação, a função de leitura de avaliações é bem-vista pela maioria dos alunos, pois pode auxiliar na escolha de livros com base nas experiências de outros usuários. Com 7,7% de respostas “Não”, isso reforça ainda mais a necessidade dessa função. O desenvolvimento e a implementação dela se torna necessária no aplicativo, uma vez que é requisitada por uma grande parte dos usuários.

Resposta pergunta 5:

Figura 5 - Pergunta 5 Questionário



FONTE: AUTORIA PRÓPRIA (2025)

O gráfico da Figura 5 demonstra as dificuldades enfrentadas pelos alunos para localizar livros específicos na biblioteca física.

Resposta – 64,1% dos entrevistados relataram dificuldades, enquanto 35,9% afirmaram não ter problemas. O dado reforça a necessidade de um sistema com busca eficiente e organização clara do acervo.

Resposta pergunta 6:

Figura 6 - Pergunta 6 Questionário



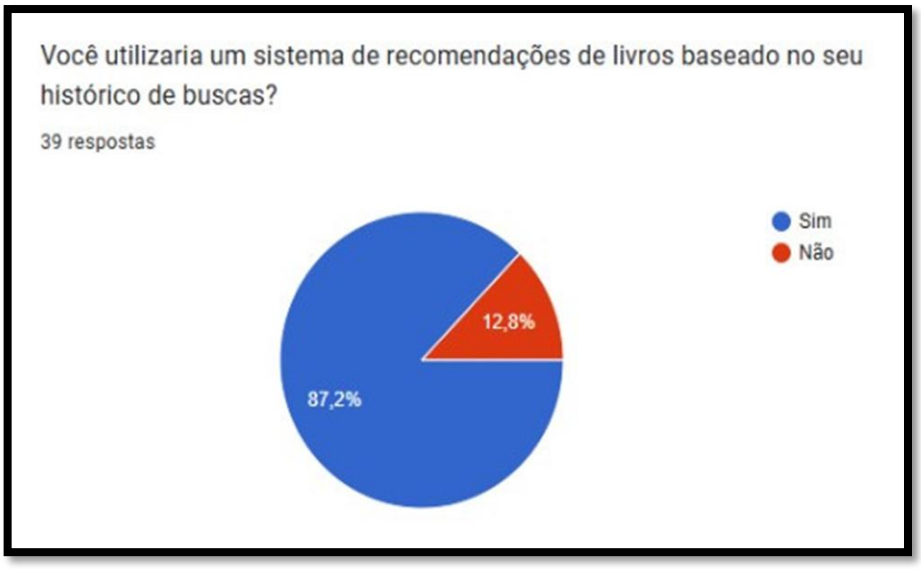
FONTE: AUTORIA PRÓPRIA (2025)

A Figura 6 mostra a opinião dos alunos quanto à importância de verificar *online* se o livro desejado está disponível.

Resposta – Com 97,4% dos alunos respondendo “Sim”, essa funcionalidade se mostra essencial para evitar deslocamentos desnecessários e otimizar o uso da biblioteca.

Resposta pergunta 7:

Figura 7 - Pergunta 7 Questionário



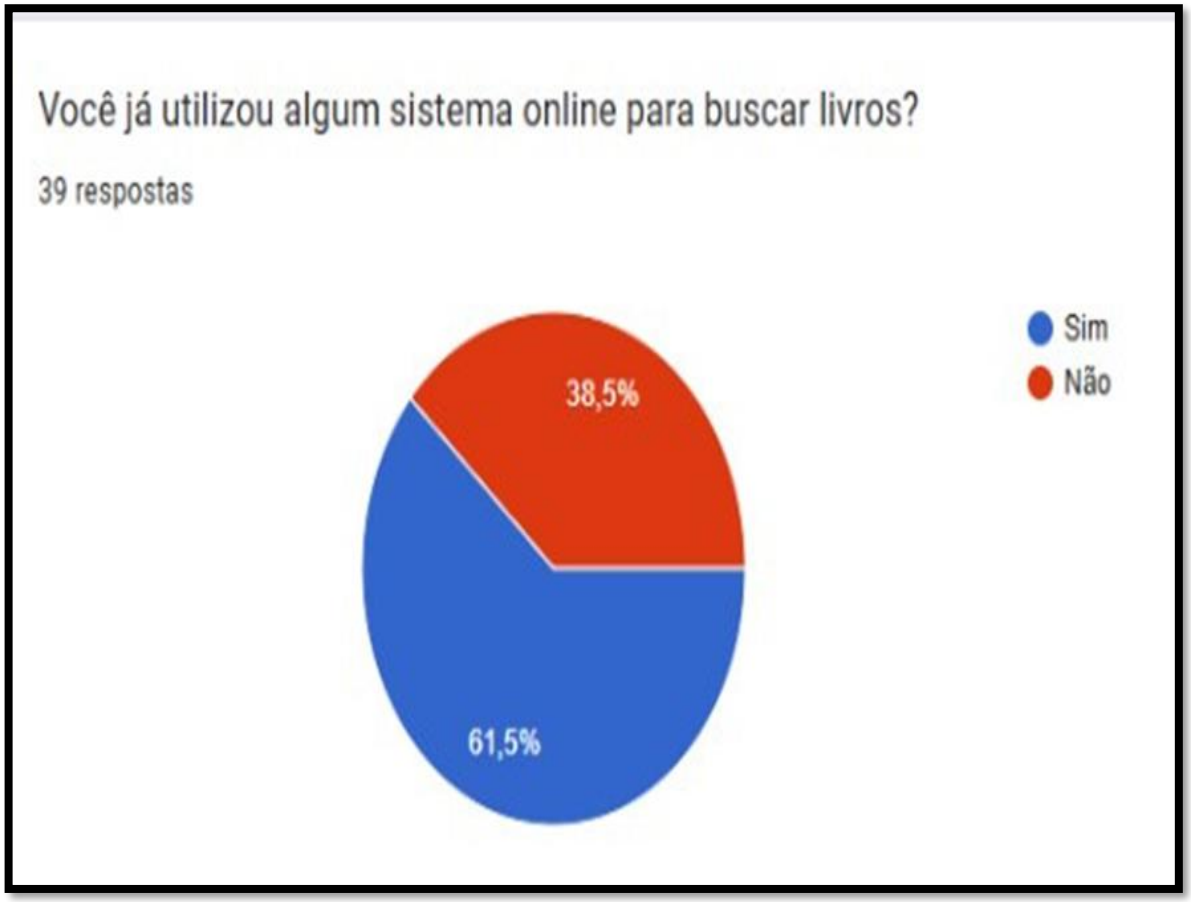
FONTE: AUTORIA PRÓPRIA (2025)

A Figura 7 apresenta as respostas sobre o uso de um sistema que sugere livros com base nos interesses e histórico de cada aluno.

Resposta – 87,2% dos alunos afirmaram que utilizariam essa função, mostrando que recomendações personalizadas são bem-vindas e podem incentivar o hábito da leitura.

Resposta pergunta 8:

Figura 8 - Pergunta 8 Questionário



FONTE: AUTORIA PRÓPRIA (2025)

A Figura 8 ilustra a familiaridade dos alunos com plataformas *online* para pesquisa de livros.

Resposta – 61,5% responderam que já utilizaram algum sistema, enquanto 38,5% não. Isso indica que muitos alunos já têm algum contato com ferramentas digitais e podem se adaptar facilmente ao novo sistema da biblioteca.

2.2.3.2 – Análise das respostas das entrevistas

Alunos:

1 – Muriel 1° Desenvolvimento de Sistemas

“Como vocês costumam procurar livros na biblioteca?”

“Quando eu fui buscar os livros eu percebi que eles estavam separados na prateleira pelo gênero literário deles, aí eu tava querendo alguns clássicos mais conhecidos então para eu pegar o livro que eu queria, eu fui na prateleira do gênero literário que eu queria e fui procurando o nome do autor na capa.”

“Quais dificuldades vocês já enfrentaram ao tentar encontrar ou pegar algum livro emprestado?”

“Acho que eu não tive, acho que foi mais a indecisão por eu ter que ficar um certo período com o mesmo livro, daí eu fico tipo ‘aí será que eu pego esse e me arrependo, será que eu deveria ter pegado outro’ só isso mesmo.”

“O que influencia na escolha do livro?”

“Eu procuro as vezes por indicação de amigos ou eu vejo em alguma rede social ou as vezes mesmo é por que é de algum livro de um gênero que eu gosto, eu pego e vejo a sinopse e por ser algo que eu já estou acostumado eu pego e vejo por conta disso mesmo.”

“Qual dificuldades você encontra ao procurar livros na biblioteca?”

“Eu acho que por conta de serem muitos títulos juntos um em cima do outro, eu acabo ficando meio perdido por as vezes perder algum livro que eu queria pegar, mas acho que é só isso da confusão mesmo que eu tenho visual.”

“Gostaria de ter acesso digital à biblioteca (*site*/app para reservas)?”

“Eu acho que seria até mais fácil assim, pois as vezes eu perco muito tempo no recreio por exemplo comendo, indo no banheiro ou terminando alguma lição e acaba não dando tempo para eu vim aqui pegar um livro que eu tinha planejado pegar, então seria legal um *site* sobre isso ou até mesmo as vezes para ver o catalogo que a biblioteca oferece.”

“O que acha da ideia de uma seção de comentários sobre os livros?”

“Eu acho que seria melhor ainda por que daí eu já pegaria o livro sabendo o que eu vou ler de fato, como se fosse uma análise do livro.”

“Se a biblioteca te indicasse livros parecidos com os que você já pegou antes, você ia curtir?”

“Também gostaria, pois como eu já falei eu gosto de pegar livros do gênero literário que eu gosto, então ficaria mais fácil ainda para mim escolher o livro.”

“Quais funcionalidades gostariam que fossem adicionadas?”

“Eu acho que além das análises e do catalogo, opções para filtrar por gêneros, autores e etc.”

2 – Emily 1° Desenvolvimento de Sistemas

“Como vocês costumam procurar livros na biblioteca?”

“Eu ainda não fui na biblioteca com intuito de buscar muitos livros, mas geralmente eu procuro principalmente pelo título.”

“Quais dificuldades vocês já enfrentaram ao tentar encontrar ou pegar algum livro emprestado?”

“Eu também não encontrei muita dificuldade para encontrar livros assim, acho que é mais a indecisão mesmo.”

“O que influencia na escolha do livro?”

“Eu costumo julgar bastante pela capa, daí eu leio bastante as sinopses para poder escolher, eu não tenho muitos amigos leitores então eu acabo não tendo essa influência.”

“Qual dificuldades você encontra ao procurar livros na biblioteca?”

“Os livros ficam muitos juntos aí acaba que eu tenho essa confusão, mas é só isso mesmo.”

“Gostaria de ter acesso digital à biblioteca (*site*/app para reservas)?”

“Eu também acho muito legal por que eu perco muito tempo procurando livro, daí eu acho que se eu tivesse em casa com o catalogo digital eu teria muito mais tempo para poder procurar.”

“O que acha da ideia de uma seção de comentários sobre os livros?”

“Eu acho uma ideia muito legal por que as vezes eu julgo bastante as notas, mas as vezes eu acho que na metade do livro eu posso acabar não gostando então seria legal.”

“Se a biblioteca te indicasse livros parecidos com os que você já pegou antes, você ia curtir?”

“Eu também acho que ficaria mais fácil não só para mim, mas como para todo mundo.”

“Quais funcionalidades gostariam que fossem adicionadas?”

“Eu não tenho muitas ideias, eu acho que essas mesmas já são legais para implementar.”

3 – Gabriel 3° Desenvolvimento de Sistemas

“Como vocês costumam procurar livros na biblioteca?”

“Eu procuro mais livros baseado no gênero, como sobre física, matemática, etc. Tem poucos livros, mas normalmente tem.”

“Quais dificuldades vocês já enfrentaram ao tentar encontrar ou pegar algum livro emprestado?”

“A maioria dos livros estão datados, velhos e deteriorando.”

“O que influencia na escolha do livro?”

“Intuição.”

“Qual dificuldades você encontra ao procurar livros na biblioteca?”

“Poucos livros sobre assuntos que me interesso, existem bastante livros sobre literatura, mas poucos na área de exatas.”

“Gostaria de ter acesso digital à biblioteca (*site*/app para reservas)?”

“Seria muito bom e excelente, pois poderia checar os livros de forma melhor e mais fácil.”

“O que acha da ideia de uma seção de comentários sobre os livros?”

“Seria bom, daria para saber a qualidade do livro e se ele é bem concorrido ou não.”

“Se a biblioteca te indicasse livros parecidos com os que você já pegou antes, você ia curtir?”

“Seria bom para descobrir mais rápido livros de meu interesse.”

“Quais funcionalidades gostariam que fossem adicionadas?”

“Reservas, filtrar categorias e checar se o livro já foi emprestado ou está reservado por outros.”

4 – Nauhany 1° Administração

“Como vocês costumam procurar livros na biblioteca?”

“Gênero.”

“Quais dificuldades vocês já enfrentaram ao tentar encontrar ou pegar algum livro emprestado?”

“Costuma demorar muito pra achar um livro pois os gêneros estão misturados em um espaço pequeno quando pego um, acaba caindo por ter dois.”

“O que influencia na escolha do livro?”

“Indicações de colegas.”

“Qual dificuldades você encontra ao procurar livros na biblioteca?”

“É que os gêneros ficam misturados e às vezes quando eu vou tirar cai mais do que um.”

“Gostaria de ter acesso digital à biblioteca (*site*/app para reservas)?”

“Sim... eu gostaria que tivesse isso, achei bem legal.”

“O que acha da ideia de uma seção de comentários sobre os livros?”

“Gostaria que tivesse.”

“Se a biblioteca te indicasse livros parecidos com os que você já pegou antes, você ia curtir?”

“Sim, achei importante.”

“Quais funcionalidades gostariam que fossem adicionadas?”

“Nossa é complicado... O tempo restante de quando você pode ficar com ele... aumentar o tempo pelo aplicativo.”

5 – Maria Eduarda 1° Desenvolvimento de Sistemas

“Como vocês costumam procurar livros na biblioteca?”

“Autor e gênero.”

“Quais dificuldades vocês já enfrentaram ao tentar encontrar ou pegar algum livro emprestado?”

“O meu maior problema é não saber as categorias da prateleira apara guardar os livros na hora de devolver.”

“O que influencia na escolha do livro?”

“Procuro opiniões de outras pessoas sobre o livro para saber sua qualidade.”

“Qual dificuldades você encontra ao procurar livros na biblioteca?”

“Nenhum, mas a impressora do Nunes para de funcionar.”

“Gostaria de ter acesso digital à biblioteca (*site*/app para reservas)?”

“Seria muito bom e útil.”

“O que acha da ideia de uma seção de comentários sobre os livros?”

“Seria bom, mas precisa de uma moderação em caso de *spoilers*.”

“Se a biblioteca te indicasse livros parecidos com os que você já pegou antes, você ia curtir?”

“Sim.”

“Quais funcionalidades gostariam que fossem adicionadas?”

“Poderia poder trocar o tamanho da fonte modo escuro e claro e poder ver a sinopse seria muito bom.”

As respostas dos alunos demonstram que há interesse em usar a biblioteca, mas muitas dificuldades atrapalham esse uso, o que reforça a necessidade de modernização. Um dos principais pontos apontados foi a organização física confusa dos livros, com títulos mal localizados, gêneros misturados e excesso de livros empilhados, o que dificulta a visualização. Vários alunos também relataram indecisão na hora de escolher um livro, por falta de informações claras ou tempo disponível nos intervalos.

Todos os entrevistados disseram que gostariam de um sistema digital, como *site* ou aplicativo, para consultar o acervo remotamente, o que tornaria a escolha e reserva de livros mais prática. Muitos destacaram que perdem tempo no recreio com outras atividades e que, se pudessem reservar antes ou de casa, seria mais eficiente. A inclusão de sinopse, capa, autor e filtros de busca (por gênero, nome, tema etc.) também foi vista como essencial para facilitar a escolha.

Outro recurso muito desejado foi uma seção de comentários ou avaliações sobre os livros, permitindo saber o que outros acharam da leitura. Também foi bem recebida a ideia do sistema sugerir livros parecidos com os que o aluno já leu, o que pode aumentar o interesse e engajamento com a leitura.

Essas informações reforçam que o projeto atende a demandas reais dos estudantes, que querem mais autonomia, praticidade e interação com a biblioteca, alinhando o espaço físico com as ferramentas digitais que já fazem parte da rotina deles.

Professoras:

1 – Maria Júlia (Professora de português)

"Como você utiliza os recursos da biblioteca nas suas aulas?"

“Inclusive hoje, eu estava organizando um projeto de leitura, por que vou pegar o livro da Malala para os primeiros anos. Então eu tento incentivar a leitura, mas como eu queria que tivesse mais títulos, ou em grande quantidade, às vezes não dá para trabalhar do jeito que eu queria. Mas acredito que essa utilização poderia ser melhor se tivesse um maior número de livros.”

“Quais dificuldades você encontra para recomendar leituras para seus alunos?”

“Principalmente a falta de interesse com as leituras. A gente pode até indicar, mas vai do interesse do aluno. A gente até fala ‘Olha agora que vocês não podem pegar o celular nos intervalos, por que não pegam um livro?’, mas eu vejo poucos com o livro na mão.”

“O que poderia facilitar o processo de indicação e reserva de livros para os alunos?”

“Uma forma deles conhecerem de uma forma mais rápida e prática os títulos existentes na biblioteca. Às vezes uma pequena resenha a respeito dos livros... Existem alguns títulos, mas quais podem interessar, qual a história, qual o assunto..., mas eu acho que eles visualizarem quais títulos tem ali, a capa por que a capa chama muita atenção, então eu acho que seria legal.”

“Você gostaria de um sistema que sugerisse materiais didáticos baseados no conteúdo que você ensina?”

“Seria legal. Até nos conteúdos que eu ensino e até nos conteúdos de interesse dos alunos. Então eu colocando o que eu ensino e o aluno colocando se gosta de ficção científica, ou romance. Então títulos que estivessem adequados aos gostos de cada um, ou a necessidade.”

“Você tivesse uma funcionalidade para criar listas de leitura para os alunos dentro do sistema da biblioteca, você usaria?”

“Seria legal. Até listas que estivessem atreladas às leituras obrigatórias dos vestibulares, às vezes o aluno nem sabe que essa lista existe ou são livros que a grande maioria não conhece. Então, organizar essa lista para conhecimento seria legal também.”

“Você acredita que uma biblioteca poderia ter um papel mais ativo no apoio ao ensino? Se sim, como?”

“Poderia sim, mais ligada ao incentivo à leitura mesmo. O aluno pegar os livros, se interessar por aquilo, perceber que aquilo é prazeroso, é divertido também. Então eu acho que seria legal.”

“O que você acha da ideia de uma aba exclusiva para professores indicarem livros e acompanharem os alunos?”

“Seria legal. Pegar a capa e colocar lá, livros e dividir por categorias. Então ficaria bem legal, seria como se você estivesse num *site* comprando aquele livro, então seria bom para a reserva, ver se está disponível, ou alugado, daqui um mês vai ficar disponível. Colocar a capa. Até quando nós fizermos a feira de troca de livros seria legal algo assim, colocar uma aba para eventos especiais. Por que o ano passado os livros estavam todos em cima da mesa e você ficava ali olhando. E se as pessoas já fossem colocando ali as capas que os alunos vão trazendo você poderia ver quais estão disponíveis, seria uma aba para eventos especiais, ficaria legal também.”

2 – Michelle (Professora de inglês/português)

“Como você utiliza os recursos da biblioteca nas suas aulas?”

“Estamos sempre fazendo trabalho sobre literatura, temos poucos exemplares sobre a literatura americana, mas na literatura europeia e brasileira existem tópicos que podem ser trabalhados, e pode ser usado como laboratório de pesquisa prática.”

“Quais dificuldades você encontra para recomendar leituras para seus alunos?”

“Seu interesse em livros, não saber com facilidade sobre todos os exemplares da biblioteca.”

“O que poderia facilitar o processo de indicação e reserva de livros para os alunos?”

“Devemos modernizar esta estrutura e disponibilizar em um app e podermos ter acesso a informações dos exemplares dos livros.”

“Você gostaria de um sistema que sugerisse materiais didáticos baseados no conteúdo que você ensina?”

“Gostaria da ideia.”

“Você tivesse uma funcionalidade para criar listas de leitura para os alunos dentro do sistema da biblioteca, você usaria?”

“Influenciaria os alunos a lerem mais, o que é algo muito importante.”

“Você acredita que uma biblioteca poderia ter um papel mais ativo no apoio ao ensino? Se sim, como?”

“A biblioteca é um complemento para o tempo de estudo, e com o professor sabendo do conteúdo da biblioteca ele pode expandir esta pesquisa para os estudos dos alunos.”

“O que você acha da ideia de uma aba exclusiva para professores indicarem livros e acompanharem os alunos?”

“Seria muito útil.”

3 – Edna (Professora de biologia)

"Como você utiliza os recursos da biblioteca nas suas aulas?"

“Utilizo os computadores da biblioteca quando os laboratórios estão ocupados e na disciplina de laboratório de investigação científica, levo os alunos para consultarem TCCs disponíveis na biblioteca para aprenderem sobre formatação e desenvolvimento de projetos também uso livros físicos, como “Contra a Realidade”, que foi comprado em múltiplos exemplares pela escola, para leitura e apresentação em sala de aula.”

“Quais dificuldades você encontra para recomendar leituras para seus alunos?”

“A principal dificuldade é fazer os alunos se interessarem por leituras obrigatórias. Os alunos gostam de ler, mas apenas o que interessa pessoalmente a eles.”

“O que poderia facilitar o processo de indicação e reserva de livros para os alunos?”

“Atualmente, depende da comunicação direta com o bibliotecário Nunes, informando com antecedência os livros e horários. Seria útil um sistema que formalizasse e automatizasse esse processo para maior agilidade.”

“Você gostaria de um sistema que sugerisse materiais didáticos baseados no conteúdo que você ensina?”

“Sim, seria muito útil. Não conheço todo o acervo da biblioteca e um sistema que mostrasse materiais disponíveis por área (como Biologia) facilitaria bastante.”

“Você tivesse uma funcionalidade para criar listas de leitura para os alunos dentro do sistema da biblioteca, você usaria?”

“Sim, com acesso facilitado às listas e aos materiais disponíveis, seria mais fácil indicar leituras aos alunos.”

“Você acredita que uma biblioteca poderia ter um papel mais ativo no apoio ao ensino? Se sim, como?”

“Sim. Seria bom se a biblioteca tivesse mais materiais específicos por área. Maior divulgação do acervo ajudaria os professores a aproveitarem melhor o espaço. Considero a biblioteca um espaço pedagógico importante, subutilizado atualmente por falta de estrutura.”

“O que você acha da ideia de uma aba exclusiva para professores indicarem livros e acompanharem os alunos?”

“Consulta ao acervo *online* para que os alunos saibam o que está disponível antes de ir até a escola. Reserva remota de livros e formação de lista de espera, especialmente quando o número de exemplares é limitado. Área com recomendações de professores.”

A partir das respostas dos professores, percebe-se uma grande necessidade de modernização do sistema da biblioteca, o que reforça a relevância do projeto proposto. Os docentes demonstraram interesse em ferramentas que possibilitem visualizar o acervo digitalmente, com capas, sinopses e filtros por área ou conteúdo, facilitando a indicação de livros aos alunos. Além disso, destacaram que um sistema que ofereça recomendações personalizadas, com base no conteúdo trabalhado em aula e nos interesses dos alunos, seria extremamente útil para o incentivo à leitura.

Outra funcionalidade considerada importante foi a possibilidade de criar listas de leitura diretamente no sistema, especialmente com foco nas leituras obrigatórias de vestibulares e materiais complementares por disciplina. Também foi bem recebida a ideia de uma aba exclusiva para professores, permitindo que eles acompanhem

quais livros os alunos estão lendo, façam recomendações e até participem de eventos especiais, como feiras de troca de livros, de forma organizada.

Em resumo, os professores acreditam que a biblioteca pode cumprir um papel mais ativo no ensino se contar com um sistema digital que automatize reservas, organize o acervo de forma clara e aproxime o aluno da leitura através da tecnologia, o que está diretamente alinhado com os objetivos do projeto.

4 – Daniel (Diretor):

“Qual a sua visão sobre o papel da biblioteca no processo educacional da escola?”

“Olha, eles têm uma importância, até mesmo porque muitas vezes o que você tá dando, você tá dando um processo resumido ali em sala de aula. Então, você tem a biblioteca lá que você consegue aprofundar no assunto, né? Quando o aluno interessa, ele vai aprofundar no assunto. Então, muitas vezes você não consegue dar todo o conteúdo na íntegra. Vai cortando, vai... pegando coisas mais uma parte, uma parte básica então, eu acho que a biblioteca é de extrema importância dentro do processo educacional por conta disso. O aluno, por conta, pode se aprofundar no assunto.”

“Você considera importante a criação de um sistema digital que integre alunos, professores e a biblioteca? Por que?”

“Ah, com certeza, né? Eu acho, assim, eu tenho certeza que a maioria dos alunos e professores não sabe nem o que tem lá dentro. Isso. Tá, então eu acho que integrando, onde todos possam ver. E assim, não foi por má vontade não, mas toda vez que a gente tentou integrar, ou o sistema é muito caro pra poder integrar com a *internet*, então, acaba tendo essa dificuldade, até mesmo financeira, de fazer esse suporte. Então, eu acho que se você perguntar para a maioria das professoras, eles não sabem o que tem lá dentro e o aluno também não sabe o que tem lá dentro. Então, eu acho que tendo esse acesso, conseguindo ver o que tem, eu acho que facilita muito a vida de todo mundo.”

“Que tipo de resultado ou indicadores você gostaria de acompanhar sobre o uso da biblioteca?”

“Sabe, eu acho que é uma coisa que acaba sendo manual ali, e você acaba não tendo essa informação direito, quantos alunos, que livros que estão saindo mais, quais alunos que frequentam mais a biblioteca. Ter essas informações acaba ajudando mesmo. A Michelle fez um levantamento, um tempo atrás, pra mim, de livros que poderiam ter na biblioteca até um projeto que a gente tendo dinheiro, a gente quer colocar uns livros diferentes ali e tal, então eu acho que você tendo quanto estão passando ali, quais livros que mais são, não digo alugados, mas que acaba o aluno pegando ali em consignação, a gente consegue trabalhar em cima disso. Hoje acaba sendo difícil, você tem ali uma coisa manual, quantos alunos que passaram e tal, mas não é uma coisa, eu acho que tendo esse controle a gente consegue direcionar melhor.”

“Quais seriam os principais desafios administrativos para modernizar o sistema da biblioteca?”

“Olha, hoje eu acredito que sejam os funcionários. Porquê...Você tem o Nunes que tem a formação de bibliotecário e tal, né? E à noite a gente tá com o senhor Mauricio lá porque não tem funcionário. Então assim, eu acho que a partir de trabalhar isso, como que vocês vão lidar, né? Ou como que vocês vão ensinar eles a mexer nesse sistema. Tem que ser um sistema de fácil operacionalização, porque senão a gente vai ter um problema de pessoas ali pra poder lidar com esse sistema. Como rodar ele, então pense nisso na hora que vocês estiverem fazendo. Tem que ser uma linguagem fácil, uma coisa fácil, porque senão... a gente acaba tendo problema de operacionalizar eles, não sabe, não sabe mexer. Então, antigamente tinha estagiário, hoje não temos nem estagiário mais. Então, aí tem que cadastrar esses livros, como que vai fazer o cadastro desses livros, o sistema. Então, tem ali o seu Maurício, que tem uma dificuldade com informática, né, pessoal, mas temos Nunes, né, que também vai precisar de uma explicação, como que vocês vão fazer isso. Acho que hoje o problema é mais a questão do funcionário mesmo.”

“Você acredita que a modernização da biblioteca pode impactar positivamente o desempenho dos alunos? De que forma?”

“Ah, eu acredito sim que você tendo uma modernização ali e assim, querendo ou não, a gente tem coisas ali dentro da biblioteca. Eu lembro, não sei como é que tá

lá, mas tinha CD (*Compact Disc* - Disco Compacto), tinha outras coisas que o pessoal pode se aprofundar mais. Então, eu acho que você modernizando a biblioteca, colocando isso E assim, a gente tá na era digital, então logo teremos obras digitais, eu acho que os TCCs poderiam ficar numa plataforma, alguma coisa, eu acho que tudo isso poderia contribuir muito mais com os alunos e não ter somente físico lá, lógico. Precisa ver tudo isso como que vai fazer, até mesmo por causa de direito autoral, toda aquela coisa, mas eu acho que você ter acesso ali, de repente você não precisa estar aqui, você está na sua casa e pode acessar, TCC, eu acho que isso seria um primordial."

"Como a direção poderia apoiar a capacitação dos professores e funcionários no uso de um novo sistema de biblioteca?"

"Eu acho assim, você estando com o sistema pronto, a gente tem reuniões pedagógicas, reuniões de planejamento, aí a gente pode demonstrar pra eles como que funciona, de uma forma mais resumida. E funcionários também, deixar treinando, marco horário, vocês veem, treinam eles. Então isso, com certeza, vai ter meu total apoio. Tá saindo, pode ter certeza que a gente vai dar um jeito pra que isso ocorra sim. Agora, professores teria que ser uma reunião pedagógica, uma reunião de planejamento, porque não dá pra você reunir eles aqui fora de horário, porque cada um tem um compromisso, tem aula em outra escola, é mais difícil, mas você tendo uma reunião dessa quando começa o ano ou termina, aí é mais fácil você demonstrar isso, pode ter certeza."

"A direção estaria disposta a investir ou buscar parcerias para o desenvolvimento desse sistema digital?"

"Com certeza. O lema aqui dentro do Centro Paulo Souza Assuntos é parceria. E aí o que depender disso, quem cuida mais de parceria é o Monfredini. Então aí de repente se você precisar de alguma coisa, conversar mais com ele nessa questão de parcerias e com certeza aqui a instituição sempre está buscando parceiros, porque se depender da verba que vem, muitas vezes não tem. Então a gente sempre procura parceiros. E aí o que precisar a gente vai atrás."

"O que você gostaria que tivesse no sistema para adquirir?"

“Então, eu acho que ele precisa trabalhar tanto com a questão de ter o acervo lá, o que eu tenho lá, mas também o que me preocupa muito é essa parte do TCC. Muitos querem ver o que tem, o que já foi feito por ter que fazer o TCC no curso que está terminando. Então, eu acho que se a gente tivesse isso dentro do sistema, sabe, artigos, algumas coisas que alimentasse esse sistema também, eu acho que ajudaria muito os alunos hoje o aluno tem que ir lá, ver o presencial. Só que eu falei, eu não sei da parte... precisa ver a questão da parte de direito autoral, tudo mais, e aí eu acho que o Nunes seria a pessoa mais indicada pra falar isso pra vocês, que eu realmente não sei como faria isso, mas...Hoje, a pessoa pra consultar tem que estar lá, tem que ver, às vezes tá em casa, você quer consultar, tá fazendo um grupo fora daqui, então eu acho que se a gente tivesse esse acervo, ou até mesmo livros que os próprios alunos pudessem fazer, tem lá o curso de poesia, que ele apoia, eu acho que movimentar a escola, fazer essa biblioteca virtual também junto com o sistema.”

A fala do diretor reforça a importância da biblioteca como apoio ao aprofundamento do conteúdo escolar, especialmente considerando que nem sempre o tempo em sala permite trabalhar todos os temas com profundidade. Ele destaca que muitos alunos e professores desconhecem o acervo atual, o que limita seu uso e por isso vê como essencial a criação de um sistema digital acessível, que permita visualizar os materiais disponíveis, sinopses, capas, e outros detalhes relevantes, democratizando o acesso à informação.

A necessidade de modernização e automatização também aparece com destaque, pois atualmente o controle é feito de forma manual e informal, dificultando o acompanhamento de dados como livros mais procurados, número de empréstimos e perfil dos usuários. O diretor acredita que um sistema com esses indicadores permitiria uma gestão mais estratégica da biblioteca.

Ele também reconhece os desafios operacionais, como a falta de funcionários capacitados e a rotatividade de pessoas com pouca familiaridade com tecnologia. Por isso, enfatiza que o sistema precisa ter uma interface simples, fácil de usar e que o projeto inclua capacitação dos funcionários e professores, preferencialmente durante as reuniões pedagógicas e horários de planejamento.

Além disso, o diretor valoriza a inclusão de acervos digitais, como TCCs e produções dos próprios alunos, e considera importante pensar na questão dos direitos autorais. Ele apoia a ideia de uma biblioteca virtual complementar, acessível também de casa, com recursos como reserva remota, acesso a artigos, e obras digitais.

Por fim, deixou claro que a direção está aberta a buscar parcerias para viabilizar o sistema, já que muitas vezes a verba pública é insuficiente. O projeto é bem-visto institucionalmente, desde que apresente soluções práticas e sustentáveis para as limitações reais da escola.

Bibliotecários:

1 – Nunes (Bibliotecário da ETEC de Mogi Guaçu)

“Como funciona o sistema da sua biblioteca?”

“Na biblioteca usamos o sistema Biblivre, que é gratuito. Os dados dos alunos são integrados na matrícula e atualizados anualmente. Os livros possuem códigos de barra, e o empréstimo é registrado no sistema. O aluno assina uma via e a outra fica com o livro. O controle é simples, mas eficaz. A cada dia de atraso, o aluno fica dois dias sem poder pegar novos livros.”

“Quais são os recursos que você sente falta ou gostaria que o sistema tivesse?”

“Gostaria que os alunos pudessem consultar o acervo de casa e que o sistema permitisse empréstimo por leitura biométrica, sem precisar digitar dados. Também seria bom permitir a renovação do livro remotamente, desde que ele não esteja reservado por outra pessoa.”

“Qual é a maior dificuldade que você enfrenta com o sistema atualmente?”

“A maior dificuldade é que o sistema não permite *backup* local. Se ele sair do ar ou for descontinuado, todos os dados podem ser perdidos. Isso me obriga a manter registros manuais como garantia.”

“Quais funcionalidades do sistema atual você mais utiliza e por quê?”

“Uso principalmente o cadastro de alunos, empréstimos, devoluções e consulta ao acervo. Mas a consulta é bem rústica e não mostra, por exemplo, onde o livro está na estante.”

“Você já teve problemas de usabilidade ao utilizar o sistema? Se sim, quais foram?”

“Sim. Por exemplo, a impressora frequentemente trava, e preciso de ajuda técnica para reiniciar tudo. Isso atrasa os processos e exige improvisos, como imprimir manualmente em papel rascunho.”

“Como você avalia o desempenho do sistema em termos de velocidade e confiabilidade?”

“Dou nota 5 de 10. É um sistema lento e limitado. Não permite que o aluno consulte ou reserve livros de casa, o que dificulta muito o acesso. A estrutura física também é precária.”

“Há alguma funcionalidade do sistema que você considera desnecessária ou mal aprimorada?”

“A impressão dos comprovantes poderia ser eliminada. O ideal seria que tudo fosse feito por meio digital, com biometria como registro de empréstimo e devolução.”

“Como é o processo de gerenciamento de livros, incluindo aquisição, catalogação e empréstimo? O sistema atual ajuda nesses processos?”

“O sistema não ajuda muito. Preciso improvisar bastante. Idealmente, o sistema deveria permitir cadastro automático com ISBN (*International Standard Book Number* - Número Padrão Internacional para Livros) e preencher informações como título, autor, número de páginas, ano, etc.”

“Você já encontrou falhas ou *bugs* no sistema? Com que frequência isso acontece e qual foi o impacto?”

“Sim, frequentemente. A impressora falha, e o sistema trava. Há falta de suporte técnico e de funcionalidades mais modernas. Isso obriga o uso de métodos paralelos e torna o serviço mais trabalhoso.”

2 – Cris (Bibliotecária do Senac)

“Como funciona o sistema da sua biblioteca?”

“Usamos o sistema BNWeb, que conecta todas as bibliotecas do SENAC. Cada aluno, ao ser cadastrado em uma unidade, tem acesso a todas. O sistema é dividido por funções e perfis, como bibliotecário ou atendente. A catalogação segue a CDD (Classificação Decimal de Dewey) e utiliza a ficha MARC (*Machine Readable Cataloging* - Catalogação Legível por Computador). O sistema gera etiquetas com código de barras, organiza livros por assunto e permite buscar por título, autor ou tema.”

“Quais são os recursos que você sente falta ou gostaria que o sistema tivesse?”

“Sinto falta de notificações quando há alterações no sistema. Também já tivemos problemas com perda de imagens e dados. Acredito que seria interessante ter mais controle sobre atualizações e uma base de assuntos mais rica.”

“Qual é a maior dificuldade que você enfrenta com o sistema atualmente?”

“As atualizações do sistema ocorrem sem aviso, e só descobrimos que algo mudou ao tentar usar a funcionalidade. Já tivemos problemas com campos que deixaram de funcionar, como o de fotos dos alunos.”

“Quais funcionalidades do sistema atual você mais utiliza e por quê?”

“Utilizo muito a catalogação, a geração de etiquetas, a organização do acervo por assunto e o controle dos empréstimos. O sistema também envia notificações automáticas e mantém histórico do aluno.”

“Você já teve problemas de usabilidade ao utilizar o sistema? Se sim, quais foram?”

“Sim. O sistema apresentou falhas como sumiço de dados e imagens. Além disso, a falta de aviso prévio sobre mudanças afeta o fluxo de trabalho.”

“Como você avalia o desempenho do sistema em termos de velocidade e confiabilidade?”

“No geral, é um sistema confiável e bem estruturado, mas essas falhas ocasionais de salvamento e comunicação das alterações prejudicam a experiência.”

“Há alguma funcionalidade do sistema que você considera desnecessária ou mal aprimorada?”

“Não considero funcionalidades desnecessárias, mas acredito que algumas poderiam ser mais completas, como a taxonomia de assuntos e a categorização de materiais (HQs (História em Quadrinhos), revistas, etc.).”

“Como é o processo de gerenciamento de livros, incluindo aquisição, catalogação e empréstimo? O sistema atual ajuda nesses processos?”

“Sim, o sistema facilita bastante. A ficha MARC organiza todos os dados do livro. O código de barras e a etiqueta são gerados automaticamente. Há controle de fila de reserva e envio de notificações por *e-mail*.”

“Você já encontrou falhas ou *bugs* no sistema? Com que frequência isso acontece e qual foi o impacto?”

“Sim. Já houve perda de imagens de alunos e falhas em funcionalidades. A resolução exige contato com a equipe de TI (Tecnologia da Informação) e pode levar dias, o que atrasa o trabalho e acumula tarefas.”

3 – Natália (Bibliotecária da ETEC de Mogi Mirim)

“Como funciona o sistema da sua biblioteca?”

“Na biblioteca usamos o sistema BiblioCPS, sistema fornecido pela própria instituição e usamos também uma planilha no Excel para controle. O acesso a ambos é feito apenas internamente, por enquanto. Gostaríamos que tivesse uma tabela *cutter* e um CDD que pudéssemos usar como pesquisa.”

“Quais são os recursos que você sente falta ou gostaria que o sistema tivesse?”

“Gostaríamos que eles tivessem acesso para saber quais livros temos aqui na biblioteca. A recomendação automática de livros é interessante, mas pode acarretar o problema de tirar nosso diálogo com os alunos. Fila de reserva gostaríamos de ter sim, apesar que temos na planilha e estamos sempre de olho e enviamos mensagem

por WhatsApp quando o livro fica disponível. O envio de notificações por *e-mail* não seria tão útil tendo em vista que os alunos nem acessam *e-mail*, todas nossas cobranças e avisos são enviados via WhatsApp.”

“Qual é a maior dificuldade que você enfrenta com o sistema atualmente?”

“Lentidão e controle de devoluções que seria excelente, mas é feito apenas na planilha.”

“Quais funcionalidades do sistema atual você mais utiliza e por quê?”

“Na planilha Excel, a funcionalidade mais utilizada é controle de empréstimos. O aluno vem na biblioteca escolher um livro, ao trazer no balcão eu peço o RM (Registro de Matrícula) dele, nisso a planilha já puxa o nome dele, coloco o tombo do livro e a planilha já puxa o nome do livro, a data de devolução é contada manualmente. No sistema do Centro Paulo Souza, a funcionalidade mais utilizada é cadastro de livros. Todos nossos livros estão cadastrados no sistema e o próprio sistema gera etiquetas que imprimimos e colamos nos livros.”

“Você já teve problemas de usabilidade ao utilizar o sistema? Se sim, quais foram?”

“Uma grande dificuldade é que o sistema frequentemente trava ou fica fora do ar, algo que nos impede de contar apenas com ele, fazendo assim a planilha no Excel um lugar mais seguro. Ao cadastrar livros tem a campo de colocar assuntos, mas é pobre de termos, fazendo com que as vezes sejamos muito genéricas.”

“Como você avalia o desempenho do sistema em termos de velocidade e confiabilidade?”

“O sistema costuma travar e ficar lento, só não atrasa nossos atendimentos porque são feitos pela planilha do Excel, mas atrasa nossos cadastros de livros.”

“Há alguma funcionalidade do sistema que você considera desnecessária ou mal aprimorada?”

“Todas as funcionalidades são usadas, mas algumas poderiam serem mais completas, como por exemplo o campo dos assuntos poderia ter mais diversidade de assuntos, no campo tipo de documento poderia ter as opções HQ e revista.”

“Como é o processo de gerenciamento de livros, incluindo aquisição, catalogação e empréstimo? O sistema atual ajuda nesses processos?”

“Ele gera etiquetas e códigos de barras. Infelizmente não classifica por CDD automaticamente e nem envia alertas de devolução, itens que nos ajudaria muito.”

“Você já encontrou falhas ou *bugs* no sistema? Com que frequência isso acontece e qual foi o impacto?”

“O sistema costuma dar problemas, já aconteceu de não conseguirmos acessar o sistema de um navegador, trocamos de navegador, e recentemente não estávamos conseguindo incluir livros, foi algo que tivemos que chamar o pessoal da TI, demorou uns 4 dias para resolverem o que fez nosso trabalho acumular.”

4 – Maria (Bibliotecária do Senai)

“Como funciona o sistema da sua biblioteca?”

“A Rede de Bibliotecas do SENAI-SP utiliza um sistema de gerenciamento de acervo que atende diferentes serviços: catalogação, circulação de materiais, emissão de relatórios, controle de usuários, consulta ao catálogo *online*.”

“Quais são os recursos que você sente falta ou gostaria que o sistema tivesse?”

“Acredito que será uma melhoria futura os usuários realizarem o serviço de autoatendimento (empréstimo, renovação e devolução).”

“Qual é a maior dificuldade que você enfrenta com o sistema atualmente?”

“Há uma variedade de relatórios que carecem de clareza quanto ao seu propósito.”

“Quais funcionalidades do sistema atual você mais utiliza e por quê?”

“Circulação de materiais, usuários, catalogação e relatórios, ou seja, maioria.”

“Você já teve problemas de usabilidade ao utilizar o sistema? Se sim, quais foram?”

“Alguns módulos, como catalogação e relatórios, apresentam interfaces não intuitivas e falta de documentação, o que dificulta a obtenção de resultados precisos na pesquisa, emissão ou registro de materiais.”

“Como você avalia o desempenho do sistema em termos de velocidade e confiabilidade?”

“O sistema é uma ferramenta confiável e rápida, amplamente utilizado por instituições de ensino superior, arquivos, museus e órgãos públicos em vários estados. Contudo, é importante que sejam fornecidos recursos adicionais, como treinamentos e orientações, para garantir que os usuários possam aproveitar ao máximo suas funcionalidades.”

“Há alguma funcionalidade do sistema que você considera desnecessária ou mal aprimorada?”

“Acredito que o vocabulário controlado seja um aspecto fundamental para melhorar a eficácia das pesquisas, uma vez que as buscas realizadas para catalogação e outras finalidades não estão produzindo resultados esperados, indicando uma possível falha na indexação dos registros anteriores.”

“Como é o processo de gerenciamento de livros, incluindo aquisição, catalogação e empréstimo? O sistema atual ajuda nesses processos?”

“O sistema é eficiente nos módulos mencionados, mas é importante considerar a evolução tecnológica e a necessidade de integração com outros sistemas. Isso permitiria uma indexação mais rápida e automática dos materiais, utilizando registros únicos e reduzindo a necessidade de digitar informações repetidas.”

“Você já encontrou falhas ou *bugs* no sistema? Com que frequência isso acontece e qual foi o impacto?”

“Até o momento, não apresentou *bugs* ou falhas.”

As respostas dos bibliotecários demonstram que, embora os sistemas atuais atendam minimamente às demandas básicas (como empréstimos, devoluções e catalogação), ainda há diversas limitações técnicas e operacionais que impactam o bom funcionamento da biblioteca e o acesso dos usuários.

O bibliotecário da ETEC de Mogi Guaçu relatou o uso do sistema Biblivre, que é gratuito, mas apresenta problemas como ausência de *backup* local, dificuldades técnicas frequentes, e impossibilidade de consulta ou reserva remota por parte dos

alunos. Também apontou a necessidade de uma interface mais moderna com reconhecimento biométrico, renovação *online* e preenchimento automático via ISBN, além de eliminar a dependência de impressão física.

Já as bibliotecárias do Senac, SENAI e ETEC de Mogi Mirim reforçaram a importância de funcionalidades como organização por assuntos, filtros de busca eficientes, envio de notificações, reserva de livros com fila de espera, e relatórios de uso mais claros. Também destacaram o valor de sistemas integrados entre unidades, autoatendimento, e acesso remoto ao acervo. Contudo, algumas enfrentam falhas recorrentes, sistemas lentos, interface pouco intuitiva, e dependência de planilhas externas (Excel) por falta de confiança nas plataformas fornecidas pelas instituições.

Além disso, muitos relataram a falta de suporte técnico ágil, o que gera acúmulo de trabalho e retrabalho. Houve também sugestões importantes como incluir novas categorias (como HQs e revistas), vocabulário controlado mais completo, classificação automática por CDD, e melhoria nos campos de assuntos.

2.2.3.3 – Conclusão

A análise dos dados obtidos por meio de entrevistas e questionários revela que há uma necessidade real de modernização no sistema da biblioteca. Alunos, professores e funcionários demonstraram interesse em uma plataforma digital que otimize o acesso ao acervo, permita reservas *online*, consultas de disponibilidade, avaliações e recomendações de livros.

Atualmente, muitos enfrentam dificuldades para localizar obras específicas e não têm acesso remoto ao catálogo da biblioteca, o que limita a autonomia dos usuários e reduz o potencial educativo do espaço. A criação de um sistema interativo e acessível pode não apenas tornar a biblioteca mais eficiente, mas também estimular o hábito da leitura e a participação ativa dos alunos.

Além disso, os bibliotecários entrevistados apontaram limitações nos sistemas utilizados atualmente, como lentidão, falhas técnicas e dificuldade de catalogação. Com base nessas informações, é possível propor um sistema que atenda melhor às necessidades dos profissionais, oferecendo funcionalidades modernas, como cadastro automatizado por ISBN, notificações, relatórios de uso e acesso remoto.

2.3 – PROTOTIPAGEM

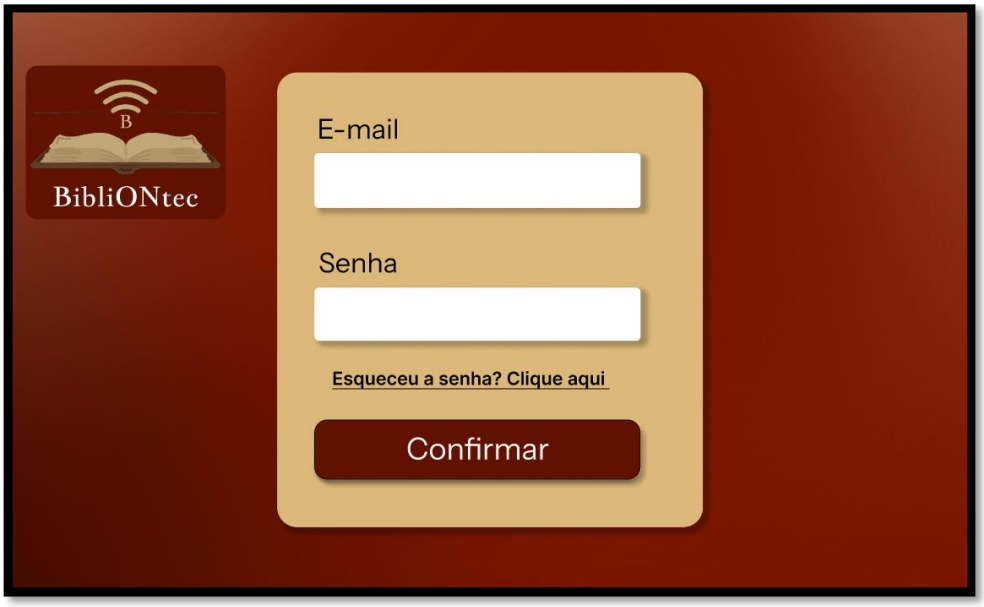
A prototipagem é um processo dinâmico de criação de um modelo inicial ou protótipo, que serve como um rascunho funcional do produto final. Essa etapa permite aos desenvolvedores testar ideias, identificar problemas e realizar melhorias antes da produção em larga escala. De maneira conceitual, a prototipagem é mais do que apenas um esboço; ela é uma exploração tangível do potencial de uma ideia, proporcionando uma visão realista de como o produto ou solução final funcionará (MIRANDA, 2023).

2.3.1 – *Site Web* – Aluno

Um protótipo de *site* é uma representação visual e interativa de um *site* em desenvolvimento. É uma ferramenta essencial para *designers* e desenvolvedores *web*, pois permite testar e validar as funcionalidades e o *design* antes da implementação final. O protótipo de *site* pode ser considerado como um esboço ou rascunho do *site*, onde é possível simular a navegação, interação com elementos e fluxo de informações (RELBERT, 2023).

2.3.1.1 - *Login*

Figura 9 - Tela *Login* Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 9 apresenta uma tela com o logotipo BibliONtec no canto superior esquerdo sobre um fundo vermelho escuro. O formulário centralizado de entrada possui fundo bege com campos para *e-mail* e senha, além de um *link* para recuperação de senha. Abaixo, um botão escuro com o texto “Confirmar” efetua o acesso ao sistema. Esta tela funciona como porta de entrada, garantindo autenticação antes do uso.

2.3.1.2 – Redefinir senha

Figura 10 - Tela Redefinir senha Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 10 apresenta uma tela com o logotipo BibliONtec no canto superior esquerdo, posicionado sobre um fundo vermelho escuro. Ao centro, um formulário com fundo bege solicita que o usuário insira seu *e-mail* para receber um código de verificação e redefinir sua senha. Abaixo do campo de *e-mail*, há um *link* com a opção "Voltar ao *login*", que redireciona o usuário. Na parte inferior do formulário, um botão marrom escuro com o texto "Mudar de senha" confirma a solicitação de recuperação.

2.3.1.3 – Confirmar redefinir senha

Figura 11 - Tela Confirmar redefinir senha Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 11 apresenta, ao centro, um formulário com fundo bege permite ao usuário redefinir sua senha, com dois campos: “Nova senha” e “Repetir senha”, garantindo que os dados sejam digitados corretamente. Abaixo, há um botão marrom escuro com o texto “Salvar”, que confirma a atualização da senha. Esta tela finaliza o processo de redefinição de senha, permitindo ao usuário atualizar suas credenciais de forma segura.

2.3.1.4 – Tela Inicial

Figura 12 - Tela Inicial Protótipo

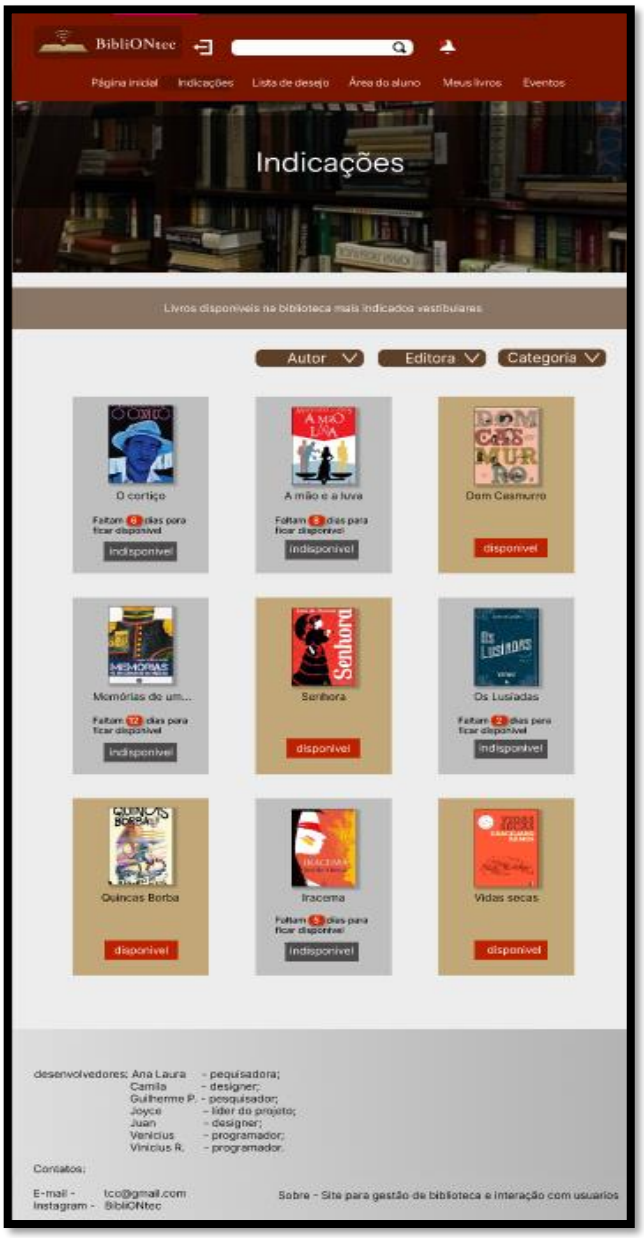


FONTE: AUTORIA PRÓPRIA (2025)

A figura 12 apresenta a tela inicial. A barra de navegação no topo inclui o logo, um campo de pesquisa e ícones de notificações e perfil. O menu horizontal oferece acesso às seções: Página inicial, Indicações, Lista de desejo, Área do aluno, Meus livros e Eventos. Abaixo, bloco de boas-vindas com explicação do *site* e destaque visual de imagem. Esta tela serve como central de navegação, proporcionando acesso rápido e visual apelativo.

2.3.1.5 – Indicações

Figura 13 - Tela Indicações Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 13 representa a tela de indicações de livros. Reaproveita o cabeçalho da página inicial. Logo abaixo, aparece o título “Indicações” sobre uma imagem de fundo com estante de livros. Abaixo, filtros disponíveis (Autor, Editora, Categoria). Em seguida, galeria de capas com *status* de disponibilidade (disponível ou dias para disponibilidade). Essa tela permite ao usuário explorar os livros recomendados, com filtragem dinâmica para melhorar a busca por materiais de interesse.

2.3.1.6 – Lista de Desejos

Figura 14 - Tela Lista de desejos Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 14 representa a tela de lista de desejos. Mantém o mesmo *layout* de cabeçalho e filtros da tela anterior. Apresenta o *banner* “Lista de desejos” e texto explicativo sobre seu funcionamento. Abaixo, exibição de capas de livros adicionados

à lista, com indicação de disponibilidade. Essa visão facilita ao usuário organizar leituras futuras e monitorar retornos de empréstimo.

2.3.1.7 – Área do aluno

Figura 15 - Tela Área do aluno Protótipo

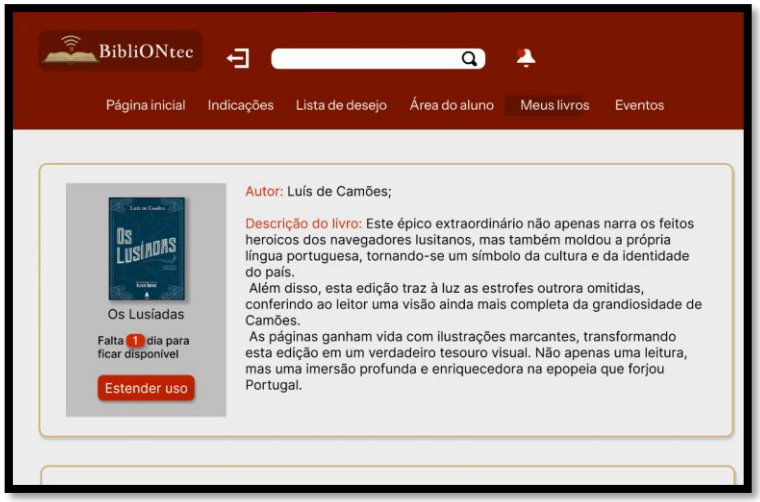


FONTE: AUTORIA PRÓPRIA (2025)

A figura 15 representa a tela da área do aluno. Cabeçalho padrão seguido por *banner* cinza-escuro com título “Área do Aluno.” Logo abaixo, painel de perfil com foto do aluno, campos para nome, *e-mail*, telefone, curso, série e senha, cada um com botão “Alterar” ao lado. A seção Notificações exibe avisos sobre prazo de devolução, com alertas visuais, *status* de situação e capas dos livros. O rodapé apresenta nomes e funções da equipe e canais de contato. Esta tela reúne informações pessoais e alertas relevantes, facilitando a gestão de perfil e comunicação personalizada.

2.3.1.8 – Meus livros

Figura 16 - Tela Meus livros Protótipo

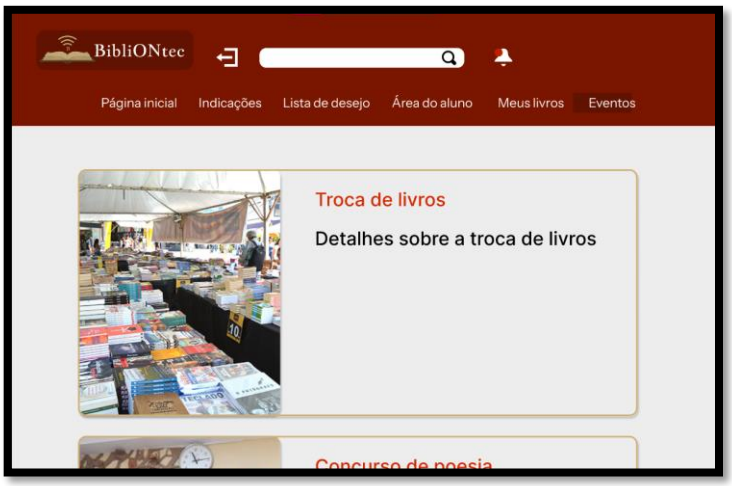


FONTE: AUTORIA PRÓPRIA (2025)

A figura 16 representa a tela de “meus livros”. Inclui cabeçalho habitual. A tela lista livros com capa, autor, descrição e *status* de disponibilidade. Livros “indisponíveis” apresentam contagem regressiva para retorno e botão “Estender uso”. Essa página permite ao usuário acompanhar empréstimos ativos, prorrogar uso, etc.

2.3.1.9 – Eventos

Figura 17 - Tela Eventos Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 17 representa a tela de “eventos”. Também com o cabeçalho padrão, apresenta lista visual de eventos como “Troca de livros” e “Concurso de poesia”, cada

um com imagem ilustrativa e breve descrição. Essa tela promove participação dos alunos em atividades culturais ligadas à biblioteca.

2.3.1.10 – Detalhes do livro

Figura 18 - Tela Detalhes do livro Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

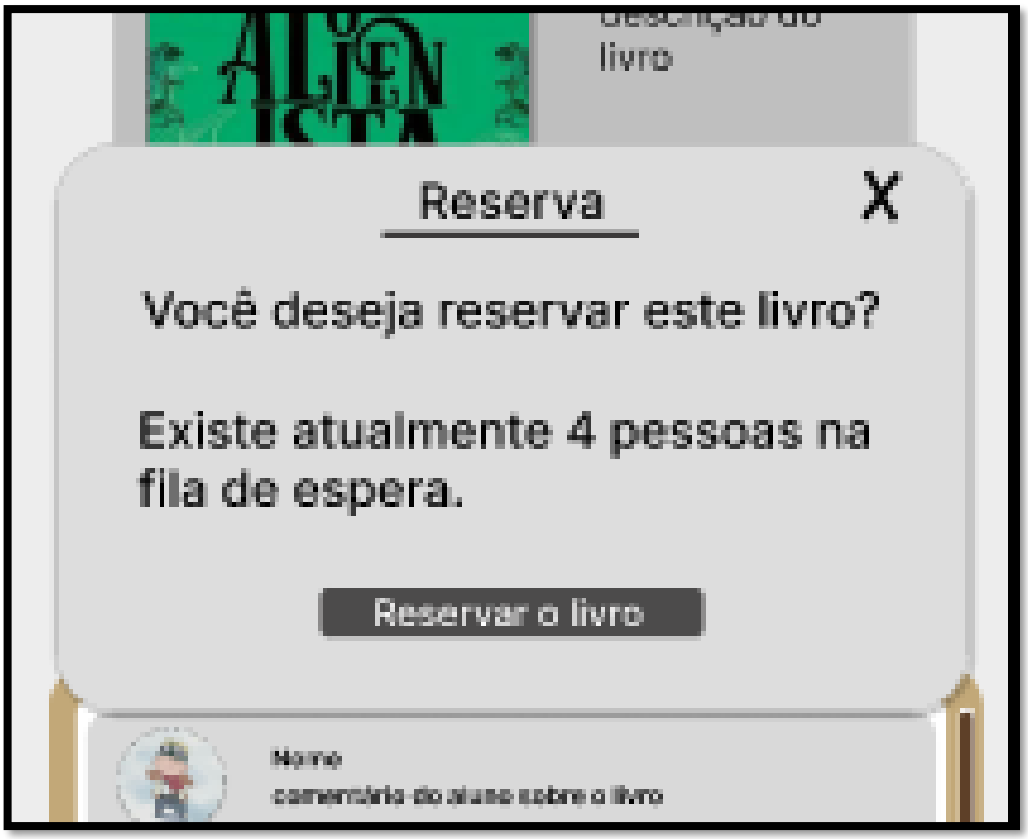
A figura 18 apresenta a tela de visualização de um livro na plataforma BibliONtec, com cabeçalho como de costume. Ao centro, é exibida a capa do livro, acompanhada de um painel com o nome, autor e uma breve descrição, além de um indicador de disponibilidade marcado como “Indisponível”.

Abaixo da descrição, dois botões são apresentados: “Reserve o livro agora!” e “Adicionar na lista de desejos”, permitindo interação com o item. Logo em seguida, uma faixa informa que faltam 6 dias para o livro ficar disponível, destacando esse número em um retângulo vermelho.

Na parte inferior da tela, uma seção exibe comentários de alunos sobre o livro, organizados em cartões com avatar, nome e comentário. Por fim, há um campo de texto para que o usuário adicione seu próprio comentário, acompanhado de um botão de envio. Esta tela fornece informações detalhadas sobre o livro e promove a interação dos usuários com avaliações e reservas.

2.3.1.11 – Reserva do livro

Figura 19 - Tela de reserva Protótipo



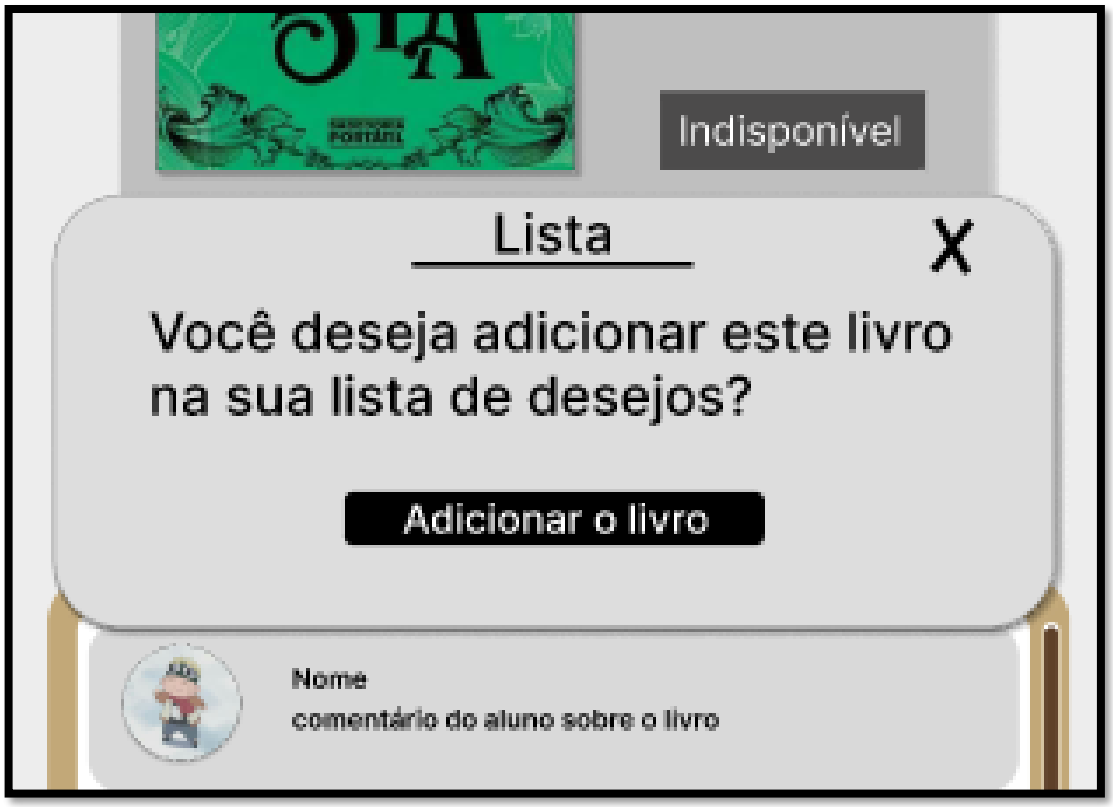
FONTE: AUTORIA PRÓPRIA (2025)

A figura 19 apresenta uma janela *modal* sobreposta à tela de detalhes do livro na plataforma BibliONtec. Com o título “Reserva” centralizado no topo, a janela exibe a pergunta “Você deseja reservar este livro?”, seguida da informação de que “Existe atualmente 4 pessoas na fila de espera.”. Abaixo, há um botão cinza com o texto

“Reservar o livro”, que permite ao usuário confirmar sua intenção de entrar na fila. No canto superior direito da janela, encontra-se um ícone de “X” que permite fechar a *modal*. Esta interface tem como função confirmar a reserva de um livro indisponível, informando previamente a posição na fila de espera.

2.3.1.12 – Adicionar livro à lista de desejos

Figura 20 - Tela Adicionar lista desejos livro Protótipo

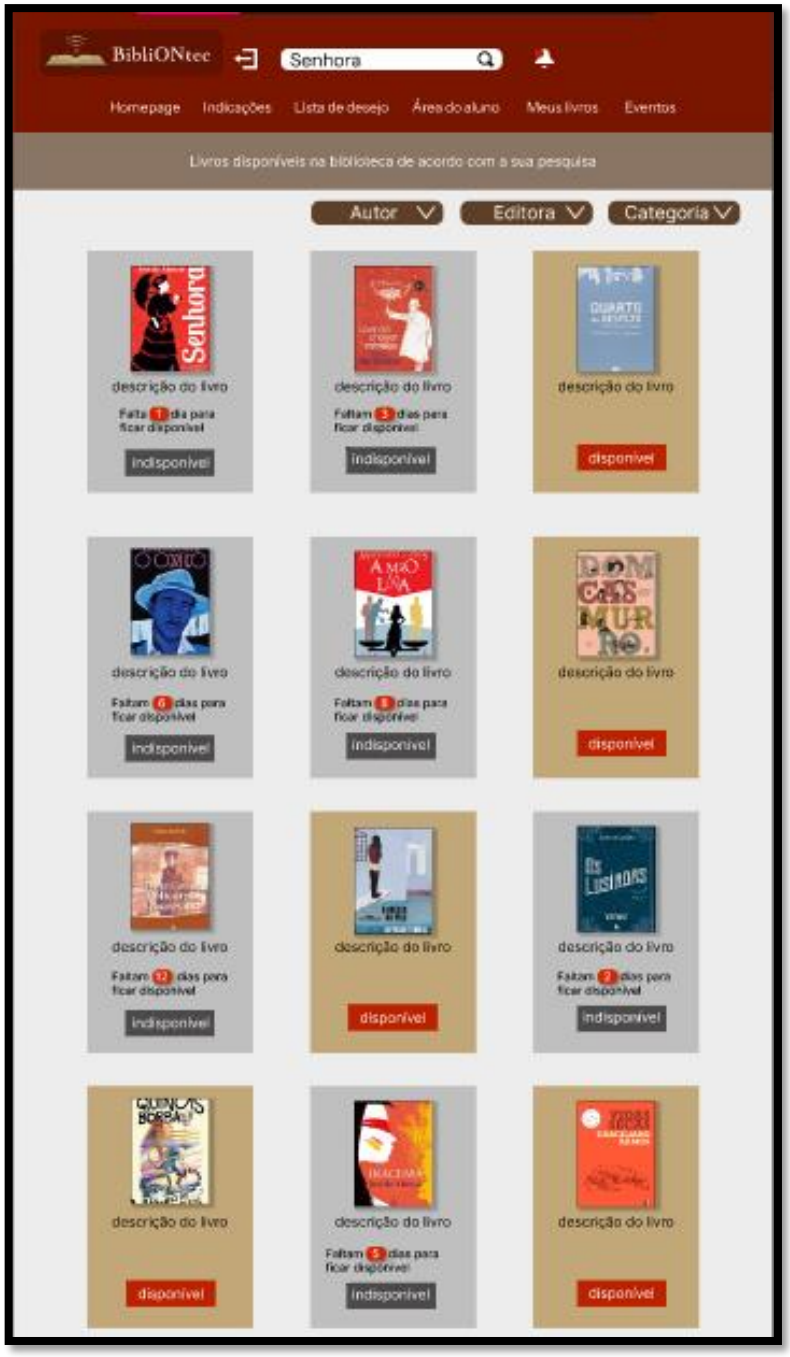


FONTE: AUTORIA PRÓPRIA (2025)

A figura 20 apresenta uma janela *modal* sobreposta à tela de detalhes do livro na plataforma BibliONtec. Com o título “Lista” centralizado no topo, a janela exibe a mensagem “Você deseja adicionar este livro na sua lista de desejos?”. Abaixo, há um botão preto com o texto “Adicionar o livro”, que permite ao usuário confirmar a ação. No canto superior direito do *modal*, um ícone de “X” possibilita o fechamento da janela. Esta interface tem como função permitir ao usuário incluir o livro selecionado em sua lista de desejos de forma prática e rápida.

2.3.1.13 – Lista de livros

Figura 21 - Tela Lista de livros Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 21 apresenta a tela de resultados de busca da plataforma BibliONtec, exibindo livros disponíveis na biblioteca de acordo com o termo pesquisado, “Senhora”. O topo da interface contém um cabeçalho vermelho com o logotipo da plataforma, barra de navegação com as opções “*Homepage*”, “Indicações”, “Lista de desejo”, “Área do aluno”, “Meus livros” e “Eventos”, além de um campo de busca.

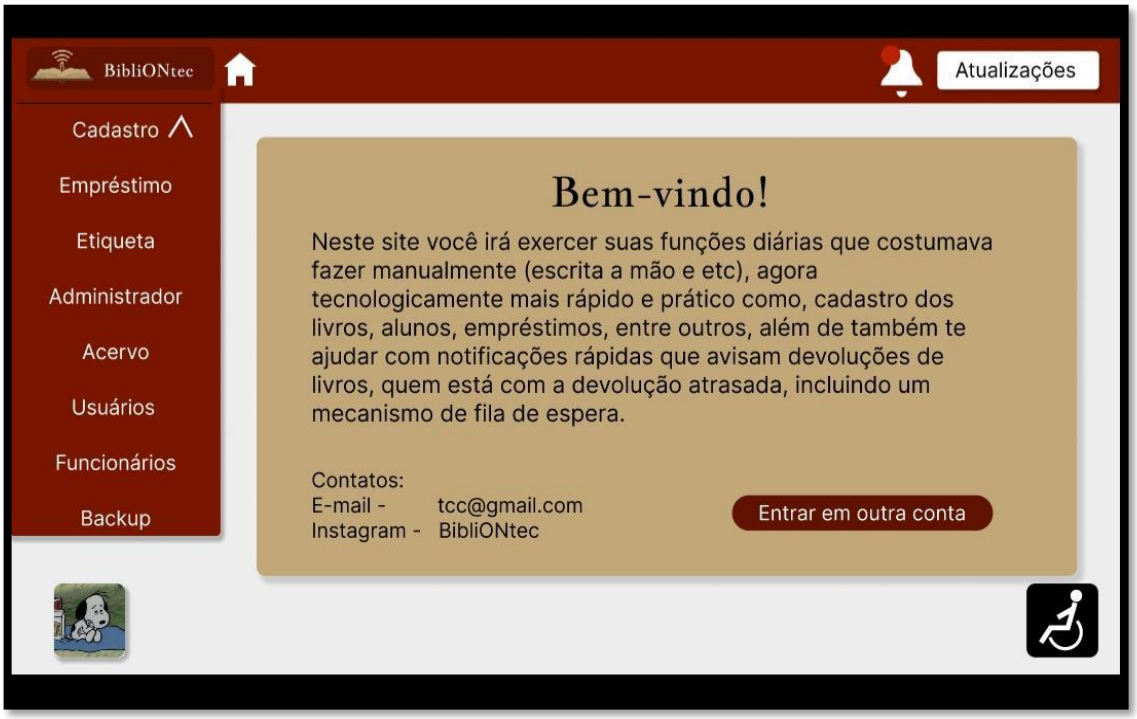
Abaixo, há filtros por Autor, Editora e Categoria, apresentados como menus suspensos em botões marrons. Em seguida, são exibidos os livros correspondentes à busca, organizados em uma grade com três colunas. Cada item possui a imagem da capa, uma breve descrição e informações sobre a disponibilidade, como o número de dias restantes para ficar disponível ou a indicação direta de “disponível” ou “indisponível”.

2.3.2 – *Site Web* – Bibliotecário

> Um protótipo de *site* é uma representação visual e interativa de um *site* em desenvolvimento. É uma ferramenta essencial para *designers* e desenvolvedores *web*, pois permite testar e validar as funcionalidades e o *design* antes da implementação final. O protótipo de *site* pode ser considerado como um esboço ou rascunho do *site*, onde é possível simular a navegação, interação com elementos e fluxo de informações (RELBERT, 2023).

2.3.2.1 – Tela Inicial

Figura 22 - Tela inicial bibliotecário



FONTE: AUTORIA PRÓPRIA (2025)

A figura 22 mostra a tela inicial do sistema, exibida após o lançamento do aplicativo. Essa interface apresenta uma visão geral simplificada, com opções de navegação para *login*, cadastro ou recuperação de acesso. O *design* é centrado na experiência do usuário, utilizando elementos visuais limpos e botões destacados que

facilitam a escolha da próxima ação. No rodapé, há informações adicionais como contato do suporte e versão do sistema.

2.3.2.2 – Trocar conta

Figura 23 - Tela Trocar conta



FONTE: AUTORIA PRÓPRIA (2025)

A figura 23 mostra a tela de troca de conta, que é acessada ao selecionar a opção “Entrar em outra conta.” Ela apresenta campos organizados para a entrada de *e-mail* e senha, permitindo ao usuário autenticar-se com credenciais diferentes. O *design* é funcional, com ênfase na simplicidade, garantindo que a troca de perfis seja feita de forma rápida e intuitiva. O campo de “esqueci senha” abre a tela de autenticação para a mudança de senha, como visto anteriormente na figura 10 no *login* de alunos.

2.3.2.3 – Atualizações

Figura 24 - Tela Atualizações



FONTE: AUTORIA PRÓPRIA (2025)

A figura 24 mostra a seção “Atualizações” que exibe uma lista de melhorias e novos recursos disponíveis no sistema. Cada atualização é apresentada com um título, uma breve descrição e, quando aplicável, um botão de ação para instalar ou ativar. O *layout* é organizado cronologicamente, destacando as mudanças mais recentes.

2.3.2.4 – Cadastro de alunos

Figura 25 - Tela Cadastro de alunos protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A tela inicial do fluxo de cadastro é exibida ao selecionar “Cadastro”, como visto na figura 25. Ela contém campos divididos por categorias, como dados pessoais, informações de contato e detalhes adicionais, oferecendo uma experiência orientada e organizada.

2.3.2.5 – Cadastro de livros

Figura 26 - Tela Cadastro de livros Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

Na figura 26, a interface “Livro” é projetada para a consulta e gestão do acervo bibliográfico. Informações como título, autor, gênero e *status* (disponível/indisponível) são apresentadas de forma organizada, com filtros para facilitar a busca. Ações como “Emprestar” e “Reservar” estão disponíveis diretamente na tela.

2.3.2.6 – Cadastro de funcionários

Figura 27 - Tela Cadastro de funcionários Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

Na figura 27, a tela “Funcionários” organiza as informações da equipe cadastrada no sistema, exibindo dados como nome, cargo e permissões. Botões de ação permitem edição, remoção ou adição de novos perfis, com um *design* intuitivo e funcional. É possível definir as permissões do funcionário ao marcar uma das opções no momento do cadastro.

2.3.2.7 – Empréstimo

Figura 28 - Tela Empréstimo Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

Como se pode ver na figura 28, a tela “Empréstimo” é projetada para gerenciar itens emprestados, com campos para identificação do usuário e do material. Há botões para confirmar o empréstimo, definir prazos de devolução e visualizar o histórico do item, promovendo controle e organização. É possível ver e gerenciar a quantidade de pessoas na fila, assim como o empréstimo e a devolução a elas.

2.3.2.8 – Etiqueta

Figura 29 - Tela Etiqueta Protótipo

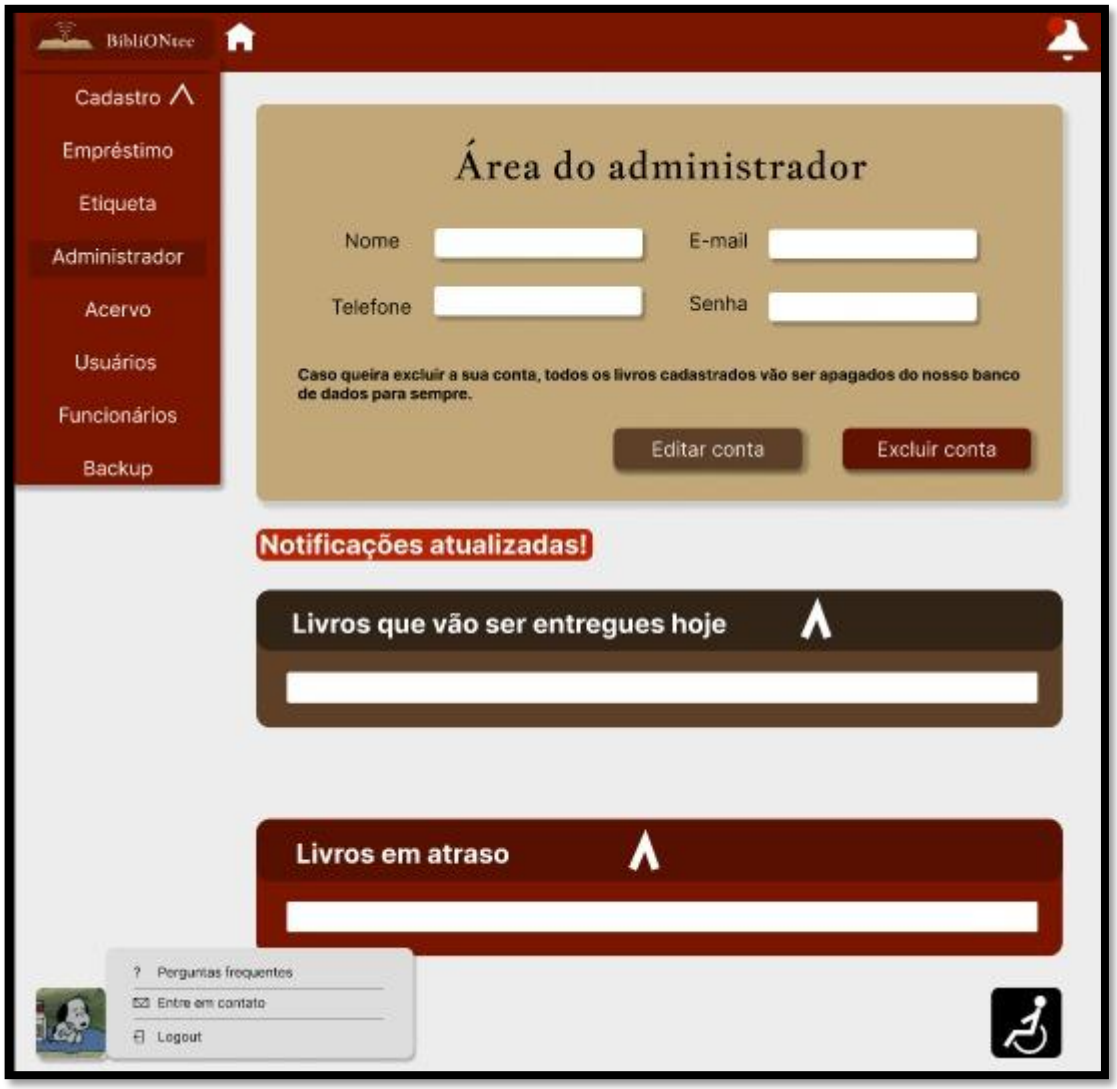


FONTE: AUTORIA PRÓPRIA (2025)

A funcionalidade “Etiqueta”, como é visto na figura 29, exibe uma interface para criar etiquetas personalizadas. O sistema oferece modelos predefinidos, além de opções para ajustar formato e conteúdo das etiquetas antes da impressão.

2.3.2.9 – Área do administrador

Figura 30 - Tela Área Administrador Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 30 mostra a funcionalidade “Administrador” abre uma tela com ferramentas avançadas de gerenciamento do sistema, incluindo configurações de permissões, geração de relatórios e controle de acessos. É possível editar informações da conta ou excluí-la e é onde se encontra notificações de empréstimos dos livros. O *design* prioriza a clareza e a eficiência no uso.

2.3.2.10 – Acervo de livros

Figura 31 - Tela Acervo Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 31 apresenta a seleção da opção “Acervo”, exibindo todos os livros disponíveis na biblioteca. O sistema também permite filtragens por Autor, Editora e Categoria, cujos ícones levam à mesma funcionalidade de refinamento da busca no acervo. Também é possível pesquisar livros pela barra de pesquisa, assim como excluí-los ou editá-los. A página torna acessível e fácil a alteração e a visibilidade dos livros presentes no acervo.

2.3.2.11 – Editar livro

Figura 32 - Tela Editar livro Protótipo

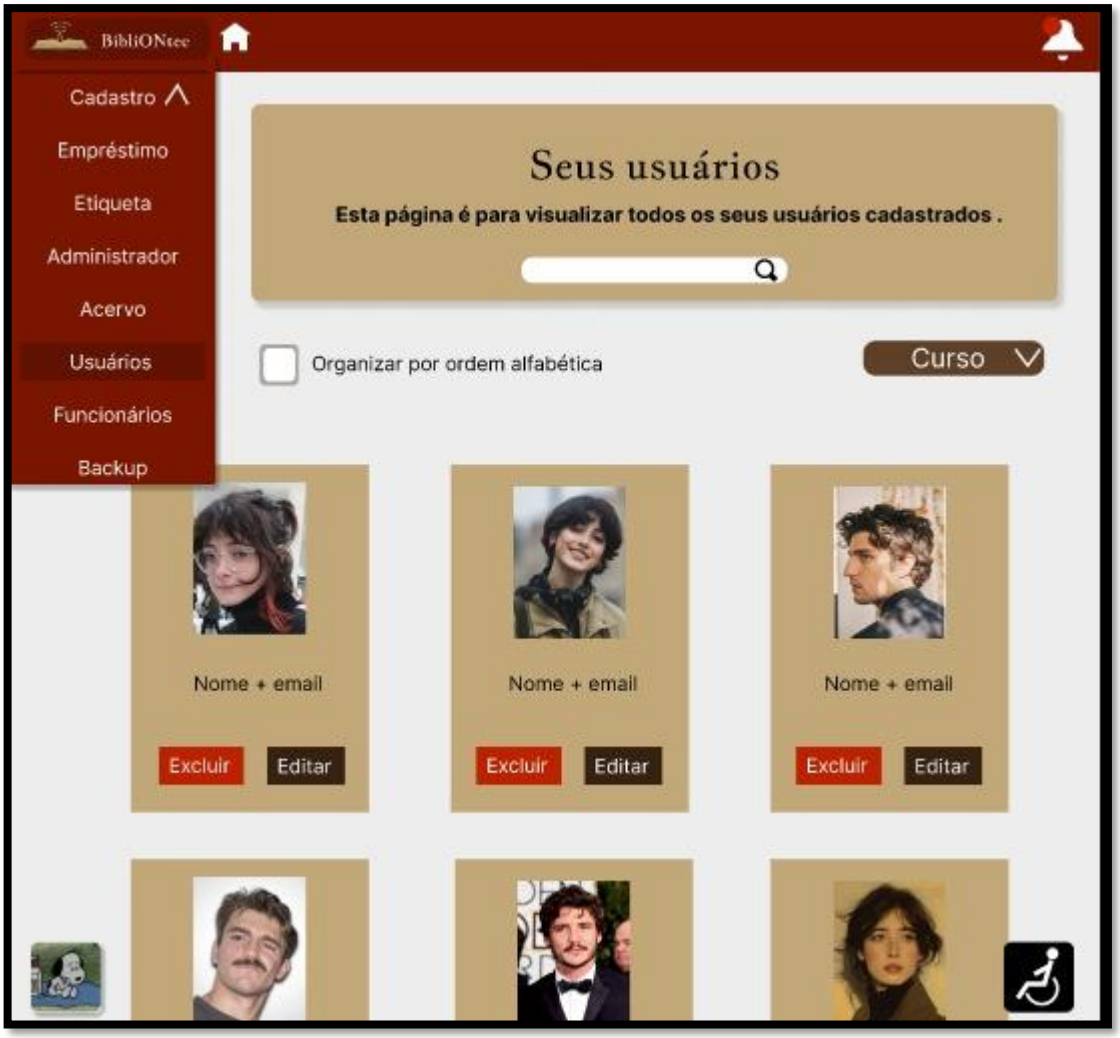


FONTE: AUTORIA PRÓPRIA (2025)

A figura 32 exibe a tela de edição de livro, ativada ao pressionar o botão “Editar livro”. A funcionalidade permite modificar informações do item no acervo, como título, autor, categoria, entre outros. Também é possível visualizar pessoas com o empréstimo desse livro no momento e seus possíveis dias de atraso. O livro também pode ser excluído por essa página, facilitando a usabilidade do sistema.

2.3.2.12 – Usuários cadastrados

Figura 33 - Tela Usuários cadastrados Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 33 mostra a visualização da seção **“**Usuários**”**, onde é possível organizar a listagem em ordem alfabética, buscar pela caixa de pesquisa ou filtrar por curso, facilitando a busca e o gerenciamento de cadastros de estudantes. É possível ver uma imagem de cada usuário, assim como seu nome e *e-mail*. Há um botão para excluir ou editar as informações de cada estudante.

Além disso, a interface oferece uma navegação intuitiva e responsiva, permitindo que o administrador acesse rapidamente os dados de qualquer usuário. O uso de filtros dinâmicos contribui para uma experiência mais eficiente, especialmente em bases de dados com grande volume de registros. A presença dos botões de ação diretamente visíveis em cada cartão de usuário agiliza a administração e reduz a

necessidade de múltiplos cliques. Essa tela é essencial para o controle de acesso dos alunos à plataforma, funcionando como um painel central de gerenciamento de perfis.

2.3.2.13 – Editar usuário

Figura 34 - Tela Editar Usuário Protótipo



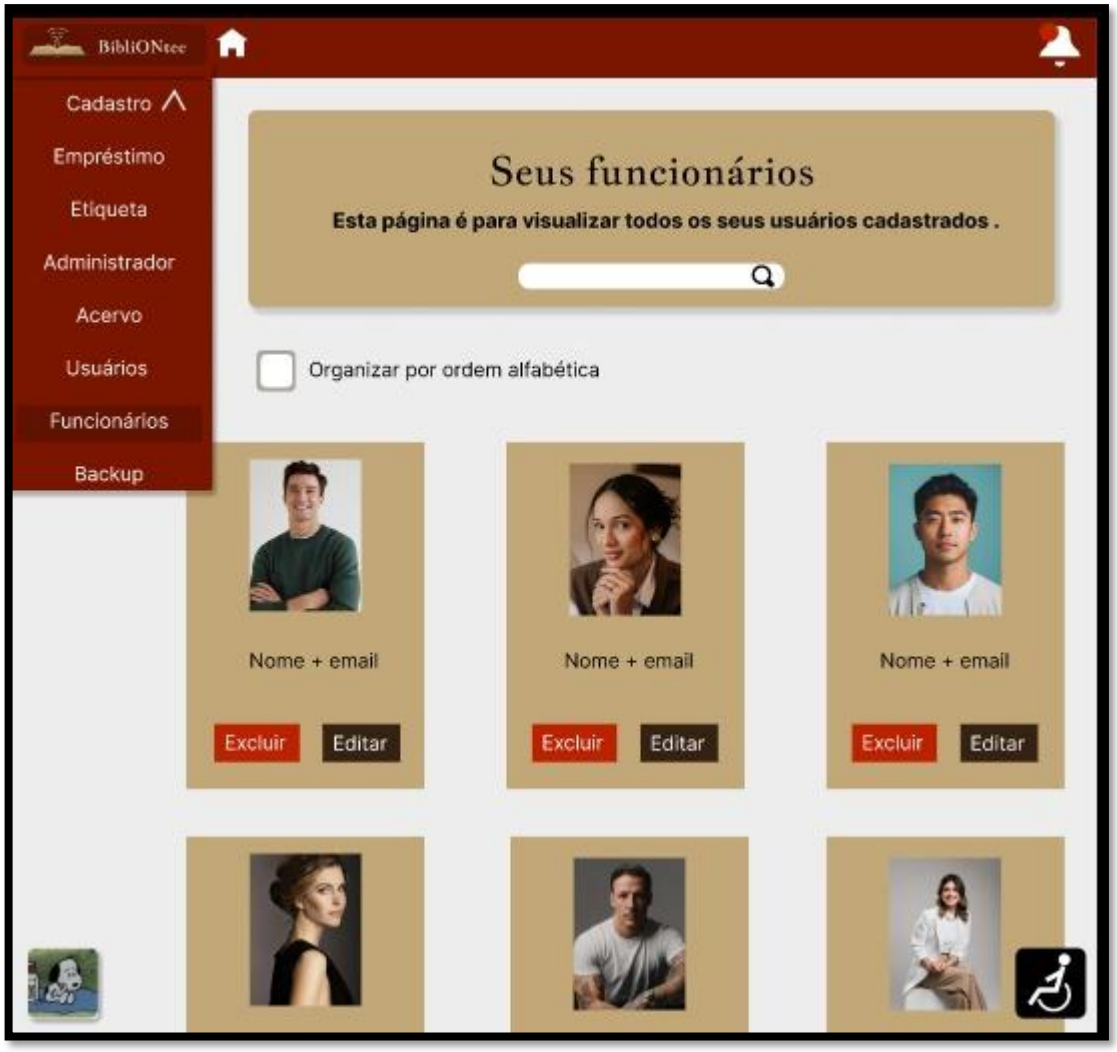
FONTE: AUTORIA PRÓPRIA (2025)

A figura 34 apresenta a funcionalidade de edição de aluno, permitindo alterar dados cadastrais e acadêmicos do estudante. Também é possível verificar a quantidade de livros sendo lidos pelo usuário no momento, assim como os seus possíveis dias de atraso. É de se notar que também é possível excluir os dados do estudante pela própria página de edição, facilitando a usabilidade e deixando mais rápido o sistema.

A interface reúne de forma clara todas as informações relevantes sobre o aluno, centralizando em uma única tela ações como edição, visualização de pendências e exclusão. Isso proporciona maior praticidade ao administrador, que não precisa navegar por múltiplas seções para acessar esses dados. Os campos de entrada são organizados de maneira lógica, permitindo uma atualização ágil e eficiente.

2.3.2.14 – Funcionários cadastrados

Figura 35 - Tela Funcionários cadastrados Protótipo

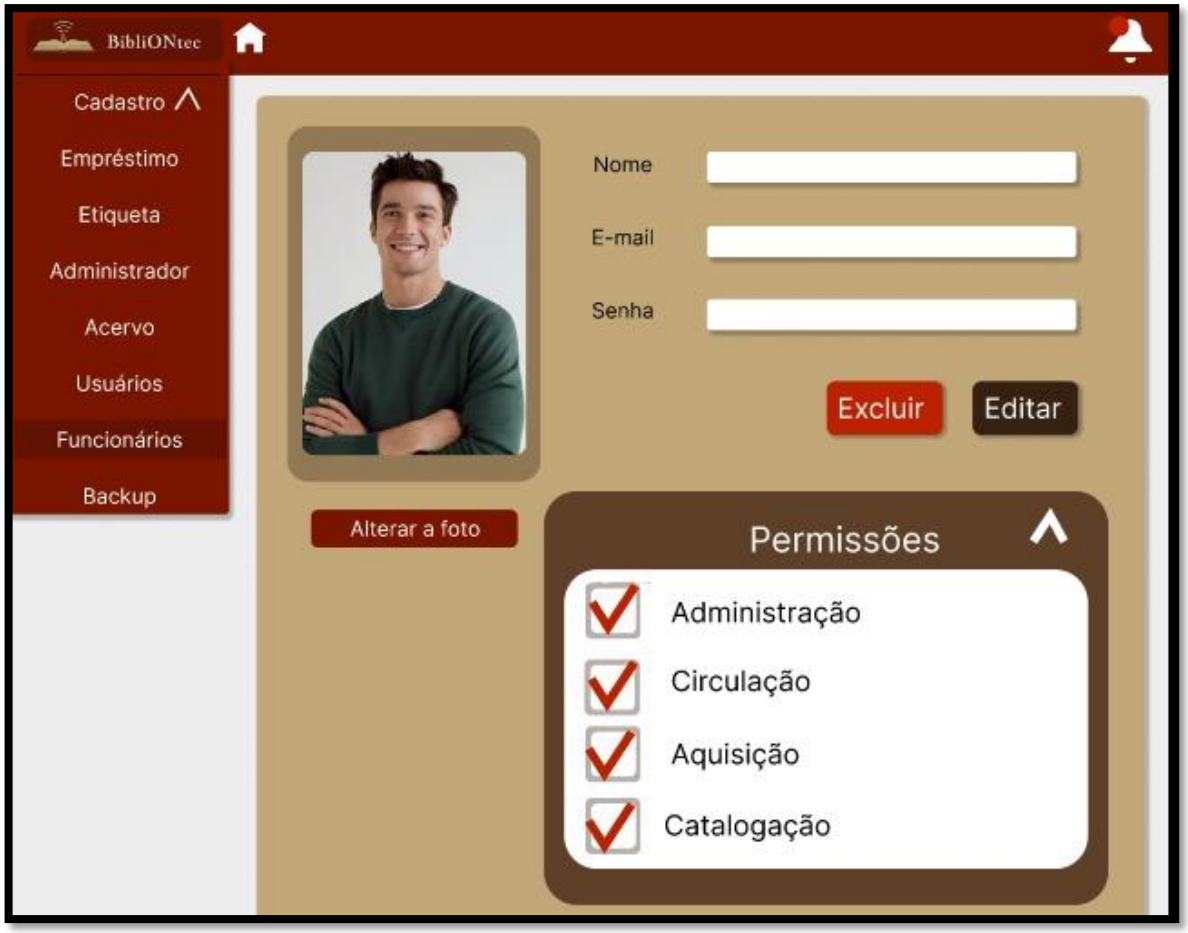


FONTE: AUTORIA PRÓPRIA (2025)

A figura 35 mostra a tela com a seleção da opção “Funcionários”, exibindo uma listagem ou gerenciamento de colaboradores do sistema. Para um gerenciamento mais fácil no sistema, é possível buscar por funcionários na caixa de pesquisa, assim como organizar por ordem alfabética. É possível ver a foto de cada funcionário cadastrado em cima de seu nome e *e-mail*. Também há um botão para excluir as informações de cada pessoa cadastrada, assim como um para editá-las, deixando o sistema mais intuitivo e fácil de usar.

2.3.2.15 – Editar funcionários

Figura 36 - Tela Editar funcionários Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

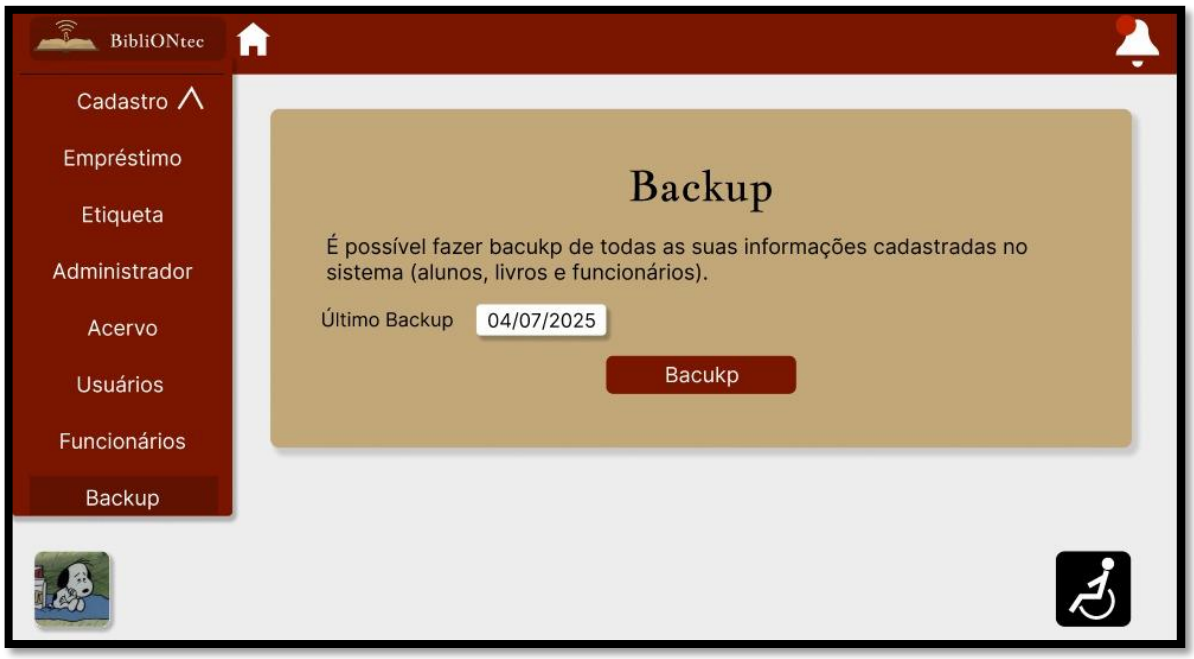
A figura 36 exibe a tela de edição dos funcionários. É possível ver e alterar os dados de cada colaborador, como: nome, *e-mail*, senha ou a sua foto de perfil. Também há a possibilidade da mudança das permissões atribuídas ao funcionário, marcando suas respectivas caixas de seleção, sendo elas: administração, circulação, aquisição e catalogação.

A organização dos campos é clara e funcional, permitindo ao administrador realizar atualizações de forma rápida e segura. Cada permissão pode ser concedida individualmente, oferecendo um controle detalhado sobre o nível de acesso de cada usuário no sistema. Essa granularidade é essencial para manter a integridade e a segurança da informação, garantindo que cada colaborador tenha acesso apenas às funcionalidades que competem à sua função. A interface ainda permite uma visualização imediata das alterações aplicadas, reforçando a transparência na gestão

interna da plataforma. Assim, o processo de manutenção do cadastro de funcionários se torna mais eficiente, reduzindo erros e otimizando o tempo da equipe administrativa.

2.3.2.16 – *Backup*

Figura 37 - Tela *Backup* Protótipo



FONTE: AUTORIA PRÓPRIA (2025)

A figura 37 mostra a tela em que foi acessada a funcionalidade de *backup*, essencial para segurança dos dados da plataforma. Também é possível verificar a data do último *backup*.

2.4 – CLIENTES

Cliente é quem compra um produto ou serviço de uma empresa. Dependendo da área, o significado pode mudar. No *marketing*, por exemplo, há o cliente interno (funcionários) e o externo (quem compra). Já na área de TI, o “cliente” pode ser um computador ou programa que faz pedidos a um servidor. O cliente é quem compra. O consumidor é quem usa. Nem sempre são a mesma pessoa (MOULIÈRES, 2024).

2.4.1 – Cliente Interno

O cliente interno é a pessoa que possui vínculo com a empresa, mesmo que não diretamente no ambiente externo com os consumidores. Normalmente, são os próprios funcionários, colaboradores ou até prestadores de serviço. Eles são

fundamentais para o funcionamento da organização e, por isso, merecem atenção e estratégias específicas (ZENDESK, 2024).

Estratégias para manter os clientes internos:

Manter os colaboradores motivados e valorizados é essencial para garantir um bom desempenho da equipe e o sucesso do projeto. Com base nas pesquisas realizadas, as estratégias incluem:

- Benefícios atrativos, como vale-transporte, vale-refeição e auxílio *home office*;
- Programas de evolução salarial, seja por tempo de empresa ou por desempenho individual;
- Ofertas de cursos gratuitos voltados ao desenvolvimento profissional, como UX (*User Experience* - Experiência do Usuário), Banco de Dados e *Marketing*;
- Reconhecimento interno, por meio de ações como “funcionário do mês” e bonificações;
- Ambiente de trabalho saudável, com horários flexíveis, *feedbacks* frequentes e reuniões para escuta ativa dos colaboradores.

Essas medidas visam valorizar os profissionais, demonstrando que são parte essencial para o sucesso do sistema desenvolvido, além de contribuir para um clima organizacional positivo.

Estratégias para recuperar clientes internos:

Quando um colaborador se desliga da empresa, é importante compreender os motivos dessa decisão. A partir do diálogo e da escuta, é possível identificar falhas internas e buscar soluções que permitam, se possível, o retorno do profissional.

O objetivo principal dessas estratégias é promover melhorias no ambiente organizacional. Ao entender as críticas e sugestões dos clientes internos, a empresa tem a oportunidade de crescer, evoluir sua gestão e criar um espaço mais acolhedor e eficiente para todos.

2.4.2 – Cliente Externo

O cliente externo é aquele que consome os produtos ou serviços da empresa, ou seja, é o público final ou parceiro comercial com o qual a organização estabelece

uma relação direta de entrega de valor. A satisfação desse cliente é fundamental para o sucesso e a reputação da empresa no mercado (ZENDESK, 2024).

Estratégias para manter os clientes externos:

- Garantir a confiança e a continuidade do relacionamento com os clientes externos exige organização, responsabilidade e compromisso com a qualidade. Com base nas pesquisas realizadas, foram adotadas as seguintes estratégias:
- Formalização de parcerias por meio de contratos, definindo claramente prazos, entregas, obrigações e suporte, evitando falhas de comunicação;
- Seleção de fornecedores confiáveis, com destaque para o uso de serviços de hospedagem reconhecidos pela estabilidade e segurança, como Hostinger, AWS ou Google Cloud;
- Configuração de hospedagens redundantes, garantindo que, se um servidor apresentar falhas, outro entra automaticamente em funcionamento, evitando interrupções no *site*;
- Realização de *backups* frequentes, com cópias armazenadas tanto em nuvem quanto em locais físicos, assegurando recuperação rápida em caso de falhas técnicas ou ataques cibernéticos;
- Acompanhamento dos serviços prestados, por meio de relatórios e validações, promovendo controle, clareza e transparência nos processos;
- Reuniões periódicas com parceiros, visando alinhar objetivos, identificar melhorias e antecipar ajustes necessários;
- Monitoramento contínuo da qualidade, com a possibilidade de substituição de fornecedores caso os padrões exigidos não sejam cumpridos.

Essas ações são fundamentais para construir relações sólidas, aumentar a confiabilidade da empresa e garantir a entrega de serviços de qualidade aos clientes externos.

Estratégias para recuperar clientes externos:

Caso ocorra a perda de um cliente externo, é importante realizar uma análise do motivo, buscando compreender o que levou ao rompimento do vínculo. A partir disso, podem ser tomadas medidas de melhoria e ajustes nos processos, com foco em recuperar a confiança do cliente.

Com essas estratégias, busca-se não apenas recuperar relacionamentos comerciais, mas também aprimorar o serviço prestado como um todo, fortalecendo a imagem da empresa no mercado e prevenindo novas perdas.

2.4.3 – Cliente Ativo

O cliente ativo é aquele que já adquiriu o produto ou serviço e continua utilizando a solução oferecida pela empresa de forma frequente. Esse tipo de cliente tem grande valor estratégico, pois além de garantir receita recorrente, pode se tornar um promotor da marca, contribuindo para sua credibilidade no mercado. (SILVA, 2025)

Estratégias para manter os clientes ativos:

Para manter o interesse e a fidelidade dos clientes que já utilizam o sistema, é essencial implementar ações que os valorizem e estimulem o uso contínuo da plataforma. Com base nas pesquisas, as estratégias adotadas são:

- Programa de Fidelidade Digital: após 12 meses de uso, o cliente recebe 15% de desconto na renovação do sistema, como forma de recompensa pela permanência;
- Clube de Testadores VIP (*Very Important Person* – Pessoa Muito Importante): após 24 meses como cliente, é possível participar de um grupo exclusivo que tem acesso antecipado a novidades e atualizações, testando funcionalidades antes do lançamento oficial;
- Sistema de Recompensas por Engajamento: os clientes acumulam pontuação com base no uso do sistema, como número de empréstimos registrados ou usuários atendidos. Esses pontos podem ser trocados por brindes e vantagens exclusivas, incentivando o uso ativo da plataforma.

Essas iniciativas têm como foco o fortalecimento do vínculo entre o cliente e a empresa, estimulando a permanência no sistema e reconhecendo a importância dos usuários ativos no crescimento do projeto.

Estratégias para recuperar clientes ativos inativos:

Caso um cliente ativo reduza ou interrompa o uso do sistema, é importante identificar os fatores que levaram à queda no engajamento. A partir de campanhas de

reativação, ofertas especiais ou ajustes nas funcionalidades, é possível recuperar esse cliente.

O objetivo é manter uma base sólida de usuários fiéis, promovendo melhorias contínuas com base em suas experiências e assegurando o uso sustentável da plataforma.

2.4.4 – Cliente Inativo

O cliente inativo é aquele que já utilizou os serviços ou produtos da empresa anteriormente, mas deixou de usá-los por algum motivo. Apesar de estar momentaneamente desconectado da organização, ele ainda representa uma oportunidade de reengajamento, já que conhece o serviço e pode ser reconquistado com as estratégias certas (SILVA, 2025).

Estratégias para recuperar clientes inativos:

Para reconquistar esses clientes, é importante entender os motivos do afastamento e apresentar melhorias que possam despertar novamente o interesse pelo serviço. Com base nas pesquisas realizadas, algumas estratégias utilizadas são:

- Pesquisa de Retorno com Bônus: envio de uma pesquisa rápida (cerca de 1 minuto) com perguntas sobre o motivo do afastamento e sugestões de melhoria. Como incentivo, o cliente recebe um brinde ou bônus, como um desconto especial ou um livro gratuito ao responder;
- Apresentação de Resultados Visuais: mostrar, de forma clara e atrativa, como o sistema contribuiu para otimizar o trabalho de outras bibliotecas, utilizando gráficos, dados e comparações reais. Isso reforça a efetividade da plataforma e pode gerar interesse em voltar a utilizá-la.

Essas ações têm como foco mostrar que o cliente ainda é valorizado pela empresa e que melhorias foram implementadas desde sua saída. A intenção é criar uma nova conexão, com base em confiança, escuta ativa e demonstração de resultados.

2.4.5 – Cliente Satisfeito

O cliente satisfeito é aquele que teve uma boa experiência com o produto ou serviço, demonstrando aprovação e confiança na empresa. Esse tipo de cliente pode

se tornar um promotor espontâneo da marca, ajudando a atrair novos usuários e fortalecendo a reputação da empresa no mercado (TOTVS, 2023).

Estratégias para manter e aproveitar o cliente satisfeito:

Manter a satisfação desses clientes e envolvê-los ainda mais com a empresa é essencial para gerar fidelização, ampliar o alcance da marca e criar um relacionamento duradouro. Com base nas pesquisas realizadas, as estratégias adotadas incluem:

- Solicitação de avaliações positivas em redes sociais, *sites* especializados e eventos, para fortalecer a imagem da empresa e conquistar novos clientes;
- Acesso antecipado a novos recursos, como forma de reconhecimento e valorização da confiança do cliente na plataforma;
- Mensagens personalizadas de agradecimento e parabenizações pelo tempo de uso, reforçando o vínculo e o reconhecimento pela parceria;
- Programas de indicação, como “Indique outra biblioteca e ganhe 1 mês grátis”, incentivando o crescimento da base de usuários por meio de recomendações espontâneas;
- Comunicação constante e atenciosa, com foco em manter o cliente engajado, informado e valorizado.

Essas ações visam transformar a satisfação em fidelidade, fazendo com que o cliente continue utilizando o sistema, divulgue espontaneamente a empresa e fortaleça a imagem positiva da marca no mercado.

2.4.6 – Cliente Insatisfeito

O cliente insatisfeito é aquele que, por algum motivo, teve uma experiência negativa com o serviço ou produto oferecido. Esse tipo de situação exige atenção imediata, pois, se bem conduzida, pode ser transformada em uma nova chance de fidelização e melhoria da empresa (TOTVS, 2023).

Estratégias para lidar com clientes insatisfeitos:

Lidar com a insatisfação exige empatia, agilidade e compromisso com a melhoria contínua. Com base nas pesquisas realizadas, as principais estratégias adotadas são:

- Atendimento empático, ouvindo o cliente com atenção, respeito e sem julgamento, demonstrando real interesse em resolver o problema;
- Resolução ágil, buscando solucionar o problema em até 24 horas, sempre que possível, mostrando eficiência e respeito pelo tempo do cliente;
- Compensações simbólicas, como forma de pedido de desculpas, oferecendo bônus ou isenção da mensalidade naquele mês, quando for adequado;
- Análise e documentação do erro, para que ele seja compreendido, corrigido e transformado em uma melhoria duradoura dentro do sistema, evitando que ocorra novamente.

Essas ações visam reconquistar a confiança do cliente, mostrar que a empresa valoriza o *feedback* recebido e que está comprometida com a qualidade do serviço. Quando bem conduzido, o atendimento ao cliente insatisfeito pode transformar uma situação negativa em um diferencial competitivo.

2.5 - ÁREA DE ATUAÇÃO

> A área de atuação é a abrangência ou o campo de trabalho onde uma pessoa ou uma empresa realiza suas atividades. Ela define o tipo de trabalho ou a indústria/setor em que alguém atua (SOUZA, 2024).

2.5.1 – Microrregião

Com base na análise econômica do projeto, foi identificado um custo total mensal de R$ 29.876,21, o que corresponde a R$ 358.514,52 ao ano, considerando despesas como aluguel, *internet*, salários, hospedagem, água, energia, manutenção e limpeza, além de domínio. Esses custos incluem a divisão proporcional dos valores de investimentos únicos (*hardware*, mobília e divulgação), que foram rateados em 12 parcelas para cálculo do custo mensal, mas também contabilizados de forma integral nos custos anuais. Isso demonstra a quantidade geral de custos geradas, considerando os custos fixos, custos variáveis, custos mensais, divulgação e entre outros.

Tabela 3 - Custos

CUSTOS	MENSAL	ANUAL
Aluguel	R$ 2.000,00	R$ 24.000,00
Internet	R$ 300,00	R$ 3.600,00
Salários	R$ 20.170,22	R$ 242.042,64
Hospedagem	R$ 10,99	R$ 131,88
Água	R$ 200,00	R$ 2.400,00
Energia	R$ 900,00	R$ 10.800,00
Manutenção e limpeza	R$ 1.000,00	R$ 12.000,00
Domínio	R$ 3,33	R$ 40,00
CUSTOS ÚNICOS		
Hardware	R$	28.000,00
Mobília	R$	1.000,00
Divulgação	R$	34.500,00
	MENSAL	ANUAL
CUSTO TOTAL:	R$ 29.876,21	R$ 358.514,52

FONTE: AUTORIA PRÓPRIA (2025)

A Tabela 3 apresenta os custos relacionados à implementação e manutenção do sistema de gerenciamento de bibliotecas, divididos entre custos mensais, custos anuais e custos únicos. Entre os principais gastos mensais estão: Aluguel (R$ 2.000,00), *Internet* (R$ 300,00) e Salários (R$ 20.170,22). Também estão inclusas despesas com Hospedagem, Água, Energia, Manutenção e Limpeza, e Domínio, resultando em um custo total mensal de R$ 29.876,21, o que equivale a R$ 358.514,52 ao ano.

Na categoria de custos únicos, destacam-se os investimentos iniciais em *Hardware* (R$ 28.000,00), Mobília (R$ 1.000,00) e Divulgação (R$ 34.500,00), valores essenciais para a estruturação inicial e a promoção do sistema.

A microrregião escolhida para a fase inicial do projeto é a cidade de Curitiba, no Paraná, que se destaca por possuir a maior quantidade de bibliotecas públicas do Brasil, totalizando 184 unidades, incluindo as 171 bibliotecas da Rede Municipal de Bibliotecas Escolares e os 13 Faróis do Saber (CRB6, 2025).

Essa escolha é estratégica, pois a cidade oferece um amplo número de instituições potenciais para parcerias com o sistema. Entre as unidades que podem

ser abordadas para uma primeira implementação estão a Biblioteca Pública do Paraná, a Biblioteca Municipal Darcy Ribeiro e a Casa da Leitura Nair de Macedo, além de diversas escolas públicas da região.

Com base na previsão de 184 bibliotecas contratantes, a uma mensalidade de R$ 200,00 por unidade, o sistema deverá gerar uma receita mensal de R$ 36.800,00, alcançando uma receita anual de R$ 441.600,00. Após a dedução dos custos operacionais, o projeto apresentará um lucro líquido mensal de R$ 6.923,79 e um lucro anual estimado de R$ 83.085,48, como demonstrado na própria tabela.

Para fortalecer a adesão inicial, a estratégia de *marketing* na microrregião inclui a oferta de um mês gratuito de uso, além da distribuição de materiais promocionais físicos, como cartazes em escolas e bibliotecas, apresentando o sistema como uma solução moderna e eficiente para a gestão de acervos.

2.5.2 – Macrorregião

A estratégia de atuação na macrorregião abrange uma expansão em nível estadual e nacional, utilizando principalmente os meios digitais para promover o sistema de gerenciamento de bibliotecas. O foco principal será a divulgação em larga escala, alcançando bibliotecas de diferentes regiões do Brasil por meio da *internet*.

Para atingir o público de outras cidades e estados, o projeto contará com campanhas de *marketing* digital realizadas em redes sociais como Instagram, LinkedIn, Facebook, WhatsApp e X (antigo Twitter). Além disso, serão feitos patrocínios a canais e influenciadores digitais voltados para os segmentos de educação, história e literatura, com o objetivo de alcançar diretamente o público interessado.

Uma das ações planejadas inclui a parceria com influenciadores como:

- Professor Noslen (YouTube e Instagram) – com mais de 5,2 milhões de inscritos no YouTube e 1,2 milhão de seguidores no Instagram, com potencial para impulsionar o alcance da campanha.
- Kabook TV (Karine Leôncio) – com presença no YouTube, Instagram, TikTok e Twitch, tendo forte influência no público leitor e educacional.

- Canal "Partiu Universidade" (antigo Se Liga Nessa História) – com foco em conteúdo de história e educação, ideal para alcançar alunos e professores.

O orçamento destinado a essas ações de *marketing* e parcerias com influenciadores é de R$ 34.500,00, valor já previsto dentro dos custos únicos do projeto.

Além das campanhas digitais, será incentivado o depoimento de bibliotecas que já utilizam o sistema, como forma de *marketing* de recomendação, criando conteúdos de divulgação em que os próprios usuários demonstrem os benefícios da aplicação. Esse material servirá para gerar confiança e atrair novas instituições.

Por fim, será realizada uma análise contínua do público-alvo nas novas regiões, com o objetivo de adaptar as campanhas conforme a aceitação, visando uma expansão orgânica e sustentável para outras escolas públicas, bibliotecas municipais, universidades e até instituições privadas de ensino.

2.6 – CONCORRENTES

Concorrente é quem disputa o mesmo espaço no mercado, oferecendo produtos ou serviços semelhantes a outro negócio. No meio empresarial, essa disputa é chamada de concorrência e é importante para incentivar melhorias, inovação e um bom atendimento ao cliente. Sem concorrência, as empresas não teriam tanto motivo para se destacar ou evoluir (FONSECA, 2022).

2.6.1 – Concorrentes Diretos

Concorrente direto é aquela empresa que oferece produtos ou serviços muito parecidos com os de outra, voltados para o mesmo público e região. Um exemplo são redes de *fast food* em uma mesma cidade, disputando os mesmos clientes. Nesse tipo de concorrência, as empresas precisam se destacar com inovações, seja no produto, no preço, na qualidade ou na forma de atendimento, buscando oferecer algo diferente para atrair mais consumidores (BENNETT, 2024).

Concorrentes Diretos da BibliONtec:

Koha

O Koha é um sistema de gestão de bibliotecas de código aberto, desenvolvido e mantido por uma comunidade global de bibliotecários e desenvolvedores. É

amplamente utilizado por grandes bibliotecas internacionais e recebe atualizações frequentes, correções de *bugs* e melhorias constantes, baseadas nas necessidades apontadas pelos usuários (KOHACOMMUNITY, 2025).

Vantagens:

- Sistema robusto, seguro e altamente estável, ideal para grandes bibliotecas (KOHACOMMUNITY, 2025).
- Atualizações frequentes, graças à colaboração de uma comunidade internacional ativa (KOHASUPPORT, 2025).
- Grande flexibilidade para personalização, por ser de código aberto (KOHACOMMUNITY, 2025).
- Quando contratado através de empresas especializadas, o Koha oferece suporte em mais de 30 idiomas, incluindo português, o que amplia sua acessibilidade para diferentes países e realidades (KOHACOMMUNITY, 2025).
- Amplo histórico de uso em instituições renomadas no mundo todo, o que traz credibilidade (KOHACOMMUNITY, 2025).

Desvantagens:

- A versão gratuita está disponível apenas em inglês, dificultando o uso por bibliotecas brasileiras (ORGANIZADATOS, 2025).
- Exige conhecimento técnico para instalação, manutenção e configuração, o que demanda uma equipe especializada (ORGANIZADATOS, 2025).
- Para ter suporte técnico e hospedagem, o custo é alto (cerca de R$ 2.214,92 mensais), o que inviabiliza para pequenas instituições (STERNHOST, 2025).
- Complexidade elevada, tornando sua curva de aprendizado maior, especialmente para bibliotecas escolares e comunitárias (ORGANIZADATOS, 2025).

Como o BibliONtec supera:

BibliONtec oferece um sistema igualmente robusto, porém com interface 100% em português, acessível por qualquer navegador, sem necessidade de instalação local, com atualizações automáticas e suporte incluso, tudo por um preço muito mais acessível (R$ 200,00) e sem necessidade de conhecimentos técnicos.

Biblivre

O Biblivre é um sistema de código aberto brasileiro, criado para atender principalmente bibliotecas públicas, escolares e de pequeno a médio porte. Oferece uma versão gratuita para instalação local e uma versão paga, hospedada na nuvem, com planos de R$ 100 a R$ 300 mensais que incluem hospedagem, *backup*, suporte e atualizações (BIBLIVRE, 2025).

Vantagens:

- Interface em português, voltada para o público brasileiro (BIBLIVRE, 2025).
- Baixo custo na versão em nuvem, o que torna viável para algumas instituições (BIBLIVRE, 2025).
- A versão gratuita é útil para quem possui infraestrutura e equipe de TI própria (BIBLIVRE, 2025).
- Sistema simples e fácil de ser utilizado em ambientes menores (BIBLIVRE, 2025).

Desvantagens:

- Atualizações pouco frequentes na versão gratuita, o que pode gerar problemas de segurança e falta de melhorias (BIBLIVRECLOUD, 2025).
- Necessidade de equipe técnica para instalar, configurar e manter a versão gratuita (BIBLIVRECLOUD, 2025).
- Funciona apenas em *desktop*, sem acesso fácil por navegadores ou dispositivos móveis (BIBLIVRECLOUD, 2025).
- Menor quantidade de recursos modernos, como notificações, reservas *online* e interações com usuários (BIBLIVRECLOUD, 2025).

Como o BibliONtec supera:

BibliONtec oferece atualizações constantes, funcionamento 100% na nuvem, acesso via navegador em qualquer dispositivo, sem depender de equipe técnica local, e inclui funcionalidades que o Biblivre não possui, como: comentários nos livros, recomendações inteligentes, sistema de eventos, funcionalidades específicas para professores e recursos de acessibilidade. Tudo isso por um valor fixo e acessível.

2.6.2 – Concorrentes Indiretos

Concorrentes indiretos são empresas que atendem às mesmas necessidades dos clientes, mas com produtos diferentes. Por exemplo, uma cafeteria e uma loja de sucos disputam o mesmo público da manhã, mesmo oferecendo bebidas distintas. Essa concorrência é menos óbvia, mas pode influenciar escolhas do consumidor com o tempo, principalmente por causa de preferências pessoais ou tendências, como saúde e bem-estar. Entender os concorrentes indiretos ajuda a empresa a acompanhar o mercado, identificar oportunidades e criar estratégias mais criativas para atrair novos clientes (BENNETT, 2025).

Concorrentes Indiretos da BibliONtec

Muitas bibliotecas, especialmente as menores, utilizam ferramentas como o Microsoft Excel para organizar seus catálogos e controlar empréstimos, embora seja uma solução rápida e acessível, o Excel não é um sistema dedicado a bibliotecas e apresenta limitações importantes (WEINSTEIN, 2013).

Vantagens:

- Amplamente disponível, pois faz parte do pacote Office já presente em muitas instituições (WEINSTEIN, 2013).
- Fácil de usar, especialmente para quem já tem familiaridade com planilhas (WEINSTEIN, 2013).
- Rápida implementação, sem necessidade de configuração complexa (WEINSTEIN, 2013).
- Oferece certa flexibilidade para registros simples e anotações personalizadas (WEINSTEIN, 2013).

Desvantagens:

- Não possui controle de usuários, *login* ou permissões, o que compromete a segurança (WEINSTEIN, 2013).
- Dados podem ser facilmente excluídos, alterados ou corrompidos, acidentalmente ou de forma maliciosa (WEINSTEIN, 2013).
- Não oferece histórico automatizado de empréstimos e devoluções (WEINSTEIN, 2013).

- Não possui notificações, reservas, nem catálogo público *online* (WEINSTEIN, 2013).
- Dificulta a manutenção colaborativa, especialmente em ambientes com mais de uma pessoa trabalhando (WEINSTEIN, 2013).
- Não é escalável, comprometendo o crescimento e a organização da biblioteca a longo prazo (WEINSTEIN, 2013).

Como o BibliONtec supera:

O BibliONtec oferece tudo o que falta no Excel, como: controle de usuários com *login* e permissões, histórico completo de empréstimos, sistema de notificações, reservas, segurança dos dados, catálogo *online*, geração de etiquetas com código de barras e acesso remoto. Tudo isso de forma simples, intuitiva e profissional, eliminando totalmente os riscos e limitações do uso de planilhas.

2.6.3 – Conclusão

O BibliONtec é uma solução inovadora que supera claramente as vantagens dos sistemas concorrentes Koha e Biblivre, ao mesmo tempo em que elimina todas as suas principais desvantagens. Enquanto o Koha oferece uma ampla comunidade internacional, sistema robusto e atualizações constantes, ele apresenta complexidade elevada na implantação, custos de suporte altos e uma versão gratuita disponível apenas em inglês — fatores que dificultam seu uso e adoção por bibliotecas brasileiras pequenas e escolares. Por outro lado, o Biblivre, apesar de ser uma opção brasileira mais simples e tradicionalmente usada em bibliotecas menores, sofre com atualizações pouco frequentes, limitações de acesso via *desktop*, e uma versão gratuita que não conta com todas as funcionalidades da versão paga, o que restringe sua escalabilidade e inovação.

O BibliONtec, por outro lado, reúne as melhores qualidades desses sistemas e vai além, oferecendo uma experiência moderna, inclusiva e eficiente. Seus diferenciais incluem:

- Interface totalmente em português, acessível por qualquer navegador;
- Atualizações automáticas, frequentes e sem custos adicionais;
- Sistema inteligente de notificações, controle de usuários, histórico e reservas;

- Comentários nos livros, criando uma rede de *s* entre leitores;
- Recomendações personalizadas com base no histórico de busca e leitura do usuário;
- Página de eventos integrada para divulgação de atividades da instituição;
- Funcionalidade exclusiva para professores recomendarem livros aos alunos;
- Geração automática de etiquetas com código de barras, facilitando a organização do acervo;
- Recursos de acessibilidade, com modo de alto contraste, suporte para leitores de tela e filtros adaptados para daltônicos, promovendo a inclusão de pessoas com deficiência visual;
- Segurança e integridade dos dados, superando soluções informais como o Excel;
- Preço acessível: R$ 200,00 mensais.

Em resumo, o BibliONtec não só resolve todas as desvantagens dos concorrentes, como adiciona funcionalidades inéditas no mercado brasileiro. É a solução ideal para instituições que buscam modernizar sua biblioteca com praticidade, economia, acessibilidade e foco real na experiência do usuário.

2.7 – LEGISLAÇÃO CONTRATUAL

No Brasil, os contratos são regulamentados por leis que definem os direitos e deveres das partes envolvidas. A principal base legal é o Código Civil, que trata da formação, validade, execução e extinção dos contratos, além de estabelecer princípios como boa-fé, autonomia da vontade e função social. Em relações de consumo, o Código de Defesa do Consumidor (CDC) também se aplica, garantindo transparência, proteção contra abusos e equilíbrio nas relações entre fornecedores e consumidores. Além dessas normas gerais, há leis específicas para diferentes áreas, como a CLT (Consolidação das leis trabalhistas), que regula contratos trabalhistas. A jurisprudência — decisões dos tribunais — também orienta a interpretação e aplicação dessas leis, e os princípios jurídicos (como equidade e segurança jurídica) complementam esse processo (SARAIVAEDUCACAO, 2023).

2.7.1 – Termos e Condições de Uso do Sistema BibliONtec

Para garantir a conformidade legal e a segurança jurídica na utilização do sistema BibliONtec, é fundamental observar as legislações aplicáveis em âmbito federal, estadual e municipal. A seguir, apresentam-se as principais normas que regem os direitos e deveres da empresa desenvolvedora, da instituição contratante e dos usuários do sistema.

Legislação Federal

Lei Geral de Proteção de Dados Pessoais (LGPD) – Lei nº 13.709/2018

Trecho: "Dados pessoais são quaisquer informações relacionadas a pessoa natural identificada ou identificável."

A LGPD regulamenta o tratamento de dados pessoais, assegurando direitos aos titulares e impondo obrigações a controladores e operadores de dados. No contexto do BibliONtec, a lei exige medidas rigorosas para coleta, armazenamento, segurança e compartilhamento de informações pessoais, respeitando o princípio da transparência e garantindo o direito à privacidade.

Lei de Acesso à Informação (LAI) – Lei nº 12.527/2011

Trecho: "As informações públicas devem estar disponíveis aos cidadãos, salvo exceções previstas em lei."

Esta lei assegura o direito de acesso às informações públicas, promovendo transparência na administração pública. No uso do sistema por instituições públicas, a LAI garante que informações institucionais estejam disponíveis, respeitando exceções legais.

Estatuto da Criança e do Adolescente (ECA) – Lei nº 8.069/1990

Trecho: "É garantido o direito à privacidade e proteção dos dados pessoais das crianças e adolescentes."

Determina proteção especial aos dados pessoais de crianças e adolescentes, exigindo consentimento dos responsáveis para o tratamento dessas informações, resguardando a privacidade desse público vulnerável.

Lei de Diretrizes e Bases da Educação (LDB) – Lei nº 9.394/1996

Trecho: "As bibliotecas escolares integram as estruturas de suporte ao processo educativo."

Estabelece que as bibliotecas escolares são componentes essenciais da estrutura educacional.

Lei Brasileira de Inclusão da Pessoa com Deficiência – Lei nº 13.146/2015

Trecho: "Os *sites* devem ser acessíveis, respeitando os padrões técnicos de acessibilidade digital, como as diretrizes WCAG."

Exige que plataformas digitais, como o BibliONtec, garantam acessibilidade, respeitando padrões técnicos reconhecidos (como as diretrizes WCAG-Diretrizes de acessibilidade para o conteúdo na *web)*, para assegurar o acesso de pessoas com deficiência.

Legislação Estadual (Paraná)

Sistema Estadual de Bibliotecas Públicas – Decreto nº 1.494/1992

Trecho: "Serviços gratuitos de consulta, leitura e empréstimo, especialmente para segmentos marginalizados."

Regulamenta a coordenação e funcionamento das bibliotecas públicas no estado, promovendo acesso gratuito e incentivando a leitura para todos os cidadãos, especialmente grupos vulneráveis.

Leis do *Software* Livre – Leis nº 14.058/2003 e nº 14.195/2003

Trecho: "O Poder Público deve priorizar o uso de *software* livre."

Incentivam o uso de *software* livre nos órgãos públicos paranaenses, buscando reduzir custos e ampliar o acesso tecnológico.

Legislação Municipal (Curitiba)

Rede Municipal de Bibliotecas Escolares (RMBE) – Decreto Municipal nº 376/2007

Trecho: "Centro de recursos informacionais para apoio ao planejamento pedagógico."

Institui e regula a rede municipal de bibliotecas escolares e comunitárias, integrando esses espaços a políticas educacionais locais, e reforçando a necessidade de sistemas eficientes para suporte à gestão dessas bibliotecas.

Compromissos do Projeto BibliONtec

Com base nas legislações mencionadas, a equipe responsável pelo desenvolvimento e manutenção do sistema BibliONtec assume os seguintes compromissos:

Garantir a proteção e segurança dos dados pessoais coletados, em conformidade com a LGPD, incluindo a adoção de medidas técnicas e administrativas para evitar acessos não autorizados.

Assegurar a transparência no tratamento de dados, informando os usuários sobre a finalidade, uso e compartilhamento das informações, respeitando os direitos dos titulares.

Promover a acessibilidade digital do sistema, implementando recursos que atendam às normas da Lei Brasileira de Inclusão, garantindo o uso por pessoas com deficiência.

Atender às exigências do ECA no que tange à proteção dos dados de crianças e adolescentes, exigindo consentimento adequado e limitando o uso dessas informações.

Colaborar com as instituições públicas e privadas para garantir a conformidade com a LAI, disponibilizando informações institucionais quando aplicável.

Priorizar o uso de *software* livre e tecnologias que promovam a sustentabilidade, alinhando-se às legislações estaduais do Paraná.

Oferecer suporte, treinamentos e documentação para os usuários, garantindo o uso adequado do sistema e a conformidade legal.

Respeitar os direitos e deveres previstos nos contratos firmados com as instituições contratantes, garantindo a segurança jurídica e o cumprimento das obrigações pactuadas.

2.7.2 – Contrato de Prestação de Serviços

Neste momento, será formalmente apresentado o Contrato de Prestação de Serviços, ressaltando-se que este instrumento é amplamente amparado e protegido pelo Código Civil Brasileiro. Este Código, enquanto documento legal de suma importância, estabelece um conjunto de normas imperativas que devem ser rigorosamente observadas em qualquer contrato de natureza empresarial.

O presente Contrato foi minuciosamente elaborado em estrita conformidade com as disposições legais vigentes, assegurando, assim, a observância dos direitos e deveres de ambas as Partes contratantes. É imperioso destacar que o Código Civil Brasileiro constitui a base jurídica fundamental para a validação e execução deste contrato, fornecendo um arcabouço normativo que garante a integridade, a segurança jurídica e a equidade das cláusulas aqui pactuadas.

Desta maneira, as Partes comprometem-se a seguir fielmente todas as condições e requisitos estipulados, em estrito respeito às normativas estabelecidas pelo ordenamento jurídico brasileiro, conferindo a este Contrato plena eficácia e validade jurídica.

CONTRATO DE PRESTAÇÃO DE SERVIÇOS DE *SOFTWARE*

Entre:

Contratante: [Nome Contratante], com sede na Rua [Endereço completo], inscrita no CNPJ (Cadastro Nacional da Pessoa Jurídica) sob nº [inserir], doravante denominada "Contratante";

Contratada: Joyce Cristina da Silva, representante legal do projeto BibliONtec, residente em [Endereço completo], inscrita no CPF (Cadastro de Pessoa Física) sob nº [inserir], doravante denominada "Contratada";

Cláusula 1 – Objeto do Contrato

1.1. O presente contrato tem como objeto a prestação de serviços pela Contratada, representante do projeto BibliONtec, para o desenvolvimento, fornecimento, implantação, manutenção e suporte técnico do sistema BibliONtec, destinado à gestão de acervos bibliotecários da Contratante.

1.2. O sistema será entregue conforme as especificações técnicas descritas no Anexo I.

Cláusula 2 – Representação e Integrantes do Projeto

2.1. A Contratada atua como representante oficial do projeto BibliONtec, composto pelos seguintes integrantes:

Ana Laura Martins de Almeida

Camila de Moraes Rocha

Guilherme Pereira

Joyce Cristina da Silva (Representante legal)

Juan Andres Sanchez de Sousa

Vinícius Alberto Ramalho

2.2. A Contratada tem plenos poderes para celebrar este contrato e tomar todas as decisões necessárias para sua execução.

Cláusula 3 – Obrigações da Contratada

3.1. Desenvolver e entregar o sistema BibliONtec conforme as especificações acordadas.

3.2. Garantir a conformidade com a LGPD, LAI, ECA, LDB, Lei Brasileira de Inclusão, decretos estaduais do Paraná e municipais de Curitiba.

A cláusula 3.2 refere-se à obrigação da contratada de assegurar que o sistema esteja em conformidade com as legislações vigentes, garantindo sua legalidade e responsabilidade social. A LGPD (Lei Geral de Proteção de Dados) impõe que qualquer dado pessoal coletado ou armazenado seja protegido, tratado com base legal e acessado apenas com consentimento ou autorização legítima. A LAI (Lei de Acesso à Informação) exige que órgãos públicos garantam transparência, mas sem violar a privacidade dos usuários, o que exige equilíbrio entre acesso à informação e proteção de dados. Já o ECA (Estatuto da Criança e do Adolescente) determina a proteção integral dos dados e direitos de menores de idade, especialmente em ambientes educacionais. A LDB (Lei de Diretrizes e Bases da Educação Nacional) estabelece que tecnologias educacionais devem seguir princípios éticos e garantir a

qualidade do ensino. A Lei Brasileira de Inclusão da Pessoa com Deficiência reforça a necessidade de acessibilidade digital, exigindo que o sistema esteja adaptado a diferentes necessidades. Por fim, os decretos estaduais do Paraná e municipais de Curitiba podem conter exigências específicas sobre segurança, transparência e acessibilidade no uso de *softwares* contratados pela administração pública, devendo também ser rigorosamente observados.

3.3. Manter confidencialidade sobre os dados da Contratante.

3.4. Fornecer suporte técnico, atualizações e manuais conforme o plano descrito no Anexo II.

3.5. Implementar medidas técnicas para garantir segurança da informação.

3.6. Comunicar qualquer falha ou incidente de segurança.

3.7. Oferecer treinamento e documentação aos usuários indicados pela Contratante.

Cláusula 4 – Obrigações da Contratante

4.1. Efetuar os pagamentos conforme estabelecido no Anexo III deste contrato, no valor de R$ 200,00 (duzentos reais) mensais, durante a vigência contratual.

4.2. Utilizar o sistema exclusivamente para fins institucionais.

4.3. Fornecer dados atualizados e colaborar com a equipe desenvolvedora.

4.4. Garantir sigilo das credenciais e reportar acessos indevidos.

4.5. Assumir responsabilidade legal sobre os dados inseridos no sistema.

Cláusula 5 – Propriedade Intelectual

5.1. Todo o código-fonte, *layout*, documentação e funcionalidades do BibliONtec são de propriedade intelectual da equipe desenvolvedora.

5.2. A Contratante recebe uma licença de uso não exclusiva, por tempo determinado, nos termos do presente contrato.

Cláusula 6 – Confidencialidade

6.1. Ambas as partes comprometem-se a manter sigilo sobre todas as informações trocadas durante a execução do contrato.

Cláusula 7 – Vigência e Rescisão

7.1. O contrato terá vigência de 1 ano, podendo ser renovado por mútuo acordo.

7.2. A rescisão poderá ocorrer mediante notificação prévia de 30 dias.

7.3. O descumprimento de qualquer cláusula ensejará rescisão imediata.

Cláusula 8 – Penalidades

8.1. A parte que descumprir o contrato estará sujeita a penalidades legais, incluindo ressarcimento por perdas e danos.

Cláusula 9 – Foro

9.1. Fica eleito o foro da Comarca de Curitiba– PR para dirimir quaisquer litígios decorrentes deste contrato.

Assinam o presente instrumento em duas vias de igual teor.

Curitiba, ___ de _____ de 20__.

Joyce Cristina da Silva

Representante legal do projeto BibliONtec – Contratada

[Nome do responsável]

Representante– Contratante

Testemunhas:

1. Nome: _____ | CPF: _____.
2. Nome: _____ | CPF: _____.

3 – DOCUMENTAÇÃO

A documentação de *software* representa um recurso indispensável no desenvolvimento e manutenção do sistema BibliONtec, sendo responsável por registrar, organizar e descrever de forma clara e estruturada os componentes, funcionalidades e processos do sistema. Vai além de simples instruções: trata-se de uma base de conhecimento essencial que permite a compreensão do funcionamento do sistema tanto por parte dos desenvolvedores quanto dos usuários finais (BRITTO, 2024).

Esse tipo de documentação oferece suporte em diversas fases do projeto, desde o planejamento até a manutenção e atualização contínua. Para a equipe técnica, ela funciona como referência para o entendimento do código, integração entre módulos, correções de falhas e implementação de melhorias. Já para os usuários, garante autonomia no uso das funcionalidades, como consultas ao acervo, realização de empréstimos, devoluções, emissão de relatórios e interações por meio da área de comentários e recomendações (BRITTO, 2024).

A documentação também contribui para a clareza das decisões técnicas adotadas, oferecendo respaldo para futuras alterações e assegurando a continuidade do trabalho mesmo em caso de troca de integrantes da equipe. Ao registrar os processos, reduz-se o risco de retrabalho, erros operacionais e mal-entendidos, promovendo a padronização das práticas e facilitando a colaboração entre os membros do grupo (BRITTO, 2024).

3.1 – OMG

Para garantir clareza e padronização na documentação de *software*, é importante considerar os padrões definidos por organizações como o OMG, que desenvolve modelos amplamente utilizados no desenvolvimento de sistemas. OMG (*Object Management Group* – Grupo de Gerenciamento de Objetos) é um consórcio internacional fundado em 1989, sem fins lucrativos, voltado à padronização de tecnologias orientadas a objetos. Seu objetivo principal é criar e manter especificações técnicas que promovam a interoperabilidade entre sistemas em diversas áreas, como desenvolvimento de *software*, engenharia de sistemas, processos de negócios, saúde, finanças, entre outras (SCIENCEDIRECT, 2010).

Entre os padrões mais conhecidos desenvolvidos pelo OMG estão a UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada), usada para modelagem de sistemas; o BPMN (*Business Process Model and Notation* - Modelo e Notação de Processos de Negócio), voltada à modelagem de processos de negócio; a SysML (*Systems Modeling Language* - Linguagem de Modelagem de Sistemas), focada em engenharia de sistemas; e a arquitetura CORBA (*Common Object Request Broker Architecture* – Arquitetura Comum de Corretor de Requisições de Objetos), que permite a comunicação entre objetos distribuídos em ambientes distintos (SCIENCEDIRECT, 2010).

O processo de criação dos padrões ocorre de forma colaborativa entre as organizações-membro do grupo. Essas entidades participam ativamente de propostas, revisões, reuniões técnicas e votações. O OMG também organiza conferências anuais e desenvolve programas de certificação, como o OCUP (OMG *Certified* UML *Professional* - Profissional Certificado em UML pelo OMG), OCEB (OMG *Certified Real-time and Embedded Specialist* - Especialista Certificado em Sistemas de Tempo Real e Embarcados pelo OMG), OCRES (OMG *Certified Real-time and Embedded Specialist* - Especialista Certificado em Sistemas de Tempo Real e Embarcados pelo OMG) e OCSMP (OMG *Certified Systems Modeling Professional* - Profissional Certificado em Modelagem de Sistemas pelo OMG), que atestam o domínio técnico sobre suas especificações (SCIENCEDIRECT, 2010).

A arquitetura de referência do OMG, chamada OMA (*Object Management Architecture* - Arquitetura de Gerenciamento de Objetos), é composta por componentes como o ORB (*Object Request Broker* - Corretor de Requisições de Objetos), responsável por intermediar a comunicação entre objetos. Essa arquitetura é dividida em categorias, como serviços de objeto, facilidades comuns, interfaces de domínio e interfaces de aplicação, possibilitando a criação de aplicações complexas e distribuídas com maior padronização e integração (SCIENCEDIRECT, 2010).

Por meio da padronização, o OMG contribui para a modernização de sistemas, reduzindo riscos, custos e o tempo de desenvolvimento, além de garantir maior compatibilidade entre tecnologias utilizadas em diferentes setores (SCIENCEDIRECT, 2010).

3.1.2 – Utilização no Sistema

Para garantir que o sistema BibliONtec operasse de forma eficiente, interoperável e compatível com diferentes plataformas, foi essencial adotar as diretrizes estabelecidas pelo *Object Management Group* (OMG). A incorporação das especificações desse consórcio, especialmente por meio da *Business Process Model and Notation* (BPMN – Modelo e Notação de Processos de Negócio), possibilitou o desenvolvimento de um sistema bibliotecário com estrutura sólida, organizado e de fácil gerenciamento.

A utilização do BPMN permitiu representar de forma clara e visual os processos internos da biblioteca, facilitando o entendimento por parte da equipe e promovendo uma comunicação mais eficaz entre os envolvidos no projeto. Além disso, a padronização dos fluxos possibilita a criação de processos flexíveis e escaláveis, que podem ser continuamente ajustados conforme as necessidades da biblioteca evoluem.

Dessa forma, o sistema não apenas atende às demandas atuais de organização, empréstimos, devoluções e comunicação com os usuários, como também está preparado para acompanhar inovações futuras. A adoção dessas práticas, recomendadas pelo OMG, contribui diretamente para a eficiência operacional do BibliONtec, além de simplificar a gestão e permitir maior agilidade na resposta às exigências de modernização do ambiente educacional.

3.2 – BPMN

O BPMN, sigla para *Business Process Model and Notation* (Modelo e Notação de Processos de Negócio), é uma notação gráfica padronizada desenvolvida pela organização *Object Management Group* (OMG), voltada à modelagem de processos de negócio. Seu principal objetivo é oferecer uma representação visual clara e compreensível dos fluxos de trabalho dentro de uma organização, permitindo que analistas, desenvolvedores e gestores tenham uma linguagem comum e unificada para descrever e analisar os processos. A adoção dessa notação facilita a identificação de gargalos, a padronização de tarefas, a comunicação entre setores e a melhoria contínua dos fluxos operacionais (SANKHYA, 2025).

A versão mais amplamente utilizada atualmente é o BPMN 2.0, que introduziu melhorias significativas, como a padronização mais rígida de símbolos, inclusão de novos diagramas e maior integração com padrões complementares como a DMN (*Decision Model and Notation* – Modelo e Notação de Decisão). A notação é composta por diversos elementos visuais, como tarefas, eventos, *gateways*, fluxos de sequência, *pools*, *lanes* e artefatos. Esses elementos permitem a descrição detalhada das etapas de um processo, indicando seu início, decisões intermediárias, interações entre áreas e encerramento (SANKHYA, 2025).

As atividades representadas no BPMN são etapas de um processo que devem seguir uma ordem lógica e ser claramente sinalizadas. Quando uma atividade não possui sequência, entende-se que se trata de um subprocesso que será iniciado ou de uma etapa que já foi concluída. As atividades podem ser realizadas tanto por pessoas quanto por sistemas, devendo sempre ser representadas dentro de padrões visuais específicos que permitam rápida identificação. Entre os tipos de atividades destacam-se as tarefas de rotina automáticas, que são executadas sem intervenção humana e incluem tarefas de serviço, tarefas de regra de negócio e tarefas de *script*. Também existem as tarefas de comunicação, utilizadas para facilitar a troca de informações entre participantes internos e externos ao processo. Nesse caso, incluem-se as tarefas de envio e de recebimento de mensagens, essenciais para garantir o andamento correto do fluxo (ALMEIDA, 2016).

O fluxo de atividades de um processo modelado com BPMN baseia-se frequentemente na troca de mensagens, mas, mesmo quando não explícita, a dependência entre etapas deve ser clara no diagrama. Quando necessário, podem ser utilizados subprocessos para complementar o fluxo e garantir sua integridade. Dessa forma, o BPMN se consolida como uma ferramenta estratégica para organizações que buscam maior eficiência, transparência e controle sobre seus processos internos, permitindo a automação de atividades e a melhoria contínua da gestão (ALMEIDA, 2016).

Além disso, a utilização da notação BPMN proporciona vantagens significativas na gestão empresarial. Dentre os principais benefícios, destaca-se a comunicação clara entre setores, uma vez que os diagramas utilizam símbolos universais compreendidos por todos os envolvidos. Isso garante maior alinhamento

entre áreas distintas da organização. A versatilidade da metodologia permite sua aplicação em diferentes tipos de processos administrativos, operacionais, financeiros, entre outros, facilitando a padronização das atividades e a identificação de gargalos (TOTVS, 2023).

Outro ponto importante é a possibilidade de automação de processos, já que os elementos do BPMN ajudam a mapear quais tarefas podem ser automatizadas, reduzindo o trabalho manual e aumentando a produtividade. Além disso, os diagramas favorecem a transparência dos fluxos, possibilitando que os envolvidos acompanhem os processos com clareza e contribuam com melhorias. Por fim, o uso do BPMN também permite o acompanhamento do desempenho dos processos, servindo como base para aplicar indicadores KPIs (*Key Performance Indicators* - Indicadores-Chave de Desempenho) e monitorar a eficiência operacional da empresa (TOTVS, 2023).

3.2.1 – Aplicação no Sistema

Dentro do sistema BibliONtec, a notação BPMN pode ser utilizada para modelar, automatizar e aperfeiçoar continuamente os processos internos da biblioteca, oferecendo uma representação visual clara de todas as etapas envolvidas. Isso inclui desde a busca por um livro, a realização do empréstimo, até a devolução e a eventual renovação do item. Ao mapear esse fluxo, a notação permite identificar e otimizar atividades como a verificação da disponibilidade, o registro do empréstimo, a atualização do sistema e o envio de notificações aos usuários.

Essa modelagem contribui para reduzir falhas operacionais, aumentar a agilidade no atendimento e minimizar erros manuais, o que favorece diretamente a experiência do usuário. Além disso, os diagramas BPMN possibilitam a identificação de gargalos nos processos e o monitoramento das ações executadas, permitindo ajustes constantes conforme as demandas da instituição. Com isso, a biblioteca aprimora sua organização e reforça seu papel como centro de acesso à informação de forma moderna e eficiente.

A modelagem de processos por meio do BPMN permite que os fluxos da biblioteca como empréstimos, devoluções, reservas, controle de prazos e penalidades sejam representados de maneira clara e compreensível. Cada etapa é visualizada por

meio de diagramas padronizados, o que facilita a comunicação entre os membros da equipe, a análise das operações e a implementação de melhorias.

Com os processos devidamente modelados, é possível também pensar em automatizações futuras. Por exemplo, a verificação de prazos de devolução ou o envio automático de lembretes por *e-mail* podem ser ações programadas com base nos fluxos BPMN, melhorando ainda mais a gestão e a eficiência do sistema.

Além disso, os diagramas permitem o monitoramento contínuo da execução dos processos, possibilitando à equipe responsável pela biblioteca identificar com maior precisão os pontos de falha, atrasos ou repetição desnecessária de tarefas, promovendo um ciclo de melhoria contínua.

3.2.2.2 – Exemplo de Aplicação

Empréstimo de Livros: O processo de empréstimo é iniciado quando um usuário realiza a solicitação por meio do sistema BibliONtec. Esse fluxo é modelado em BPMN, descrevendo detalhadamente todas as etapas necessárias, como a verificação da disponibilidade do item desejado, a confirmação do cadastro do usuário no sistema e a validação de possíveis restrições ativas. Em seguida, o empréstimo é registrado digitalmente e o sistema gera automaticamente um aviso sobre a data de devolução futura, garantindo organização e maior controle das operações da biblioteca.

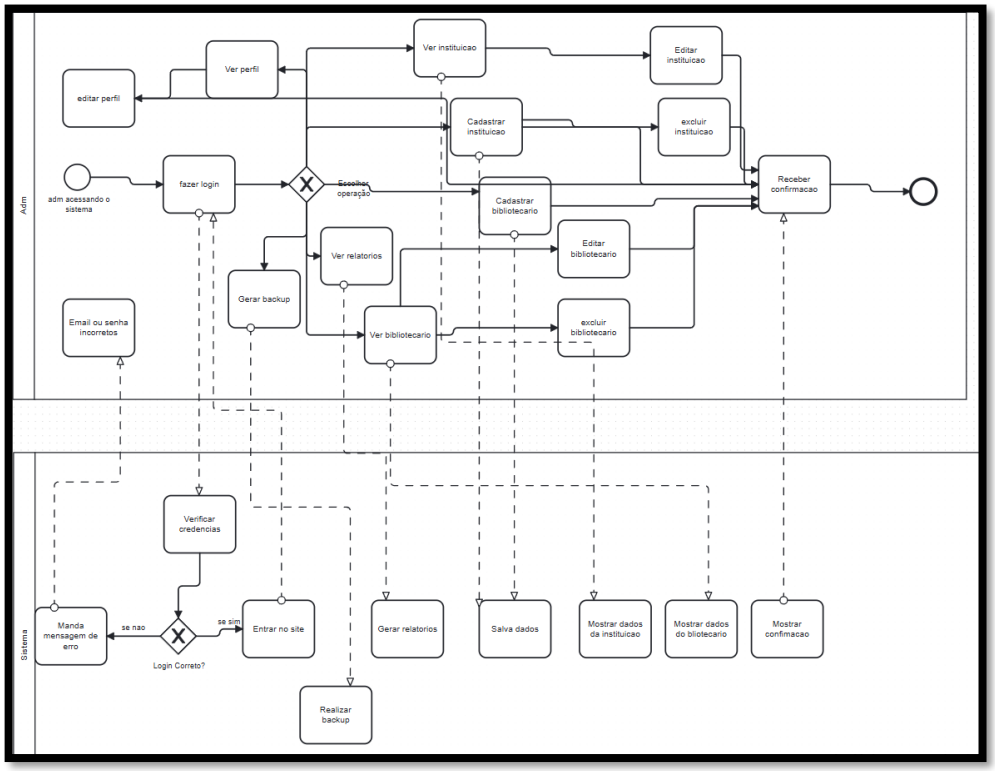
Controle de Devoluções e Penalidades: Outro processo importante que pode ser representado em BPMN é o controle de prazos e a aplicação de penalidades. Esse modelo contempla desde a contagem do prazo a partir da data do empréstimo, o envio de lembretes automáticos próximos ao vencimento, até a geração de multa em caso de atraso. Esse procedimento não apenas assegura a devolução correta dos itens, mas também contribui para a manutenção do acervo em bom estado, além de estimular a responsabilidade e o uso consciente pelos usuários.

Reserva de Itens: A modelagem em BPMN também se aplica ao processo de reserva de livros e outros materiais. Nesse fluxo, o sistema organiza ações como a solicitação de reserva pelo usuário, a análise de disponibilidade futura do item, a definição da ordem em fila de espera e a emissão de notificação automática quando o exemplar estiver disponível. Esse recurso permite uma gestão eficiente da

demanda, evitando conflitos entre usuários e assegurando maior transparência e acessibilidade no acesso ao acervo.

3.2.2 – BPMN do administrador

Figura 38 - BPMN do Administrador



FONTE: AUTORIA PRÓPRIA (2025)

O diagrama BPMN representado na Figura 38 descreve o fluxo de atividades do ator Administrador dentro do sistema BibliONtec. Seu objetivo é apresentar, de forma visual e sequencial, os passos realizados pelo administrador desde o acesso até o gerenciamento institucional e de bibliotecários.

O processo se inicia com o *login* no sistema, onde o administrador informa suas credenciais. O sistema então executa a verificação: em caso de erro, é exibida uma mensagem de falha; caso contrário, o sistema libera o acesso. A partir daí o administrador pode escolher diferentes operações.

As ações disponíveis incluem:

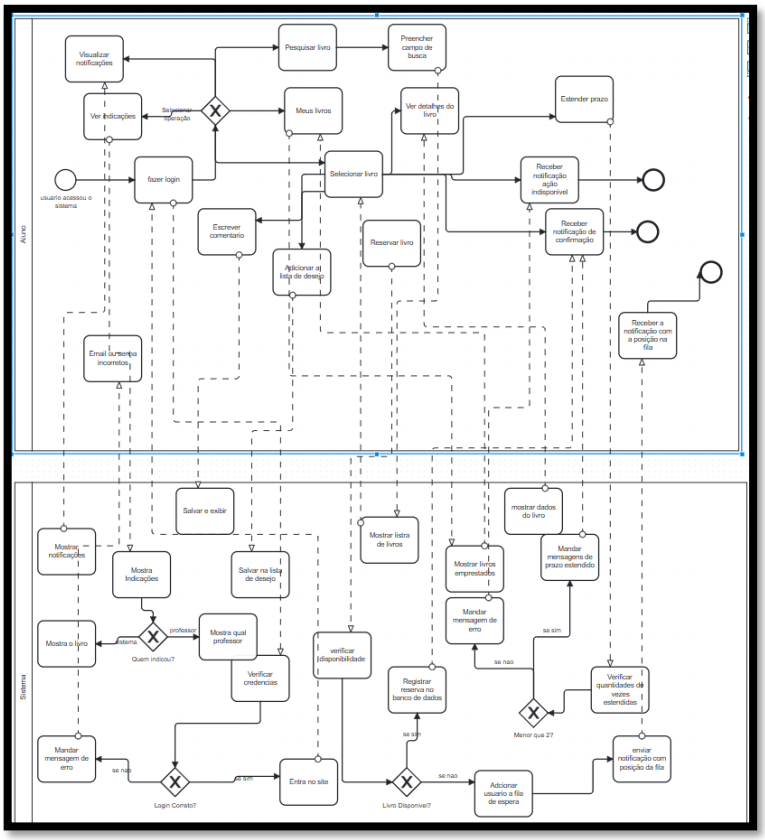
- Visualizar e editar o próprio perfil;
- Gerenciar a instituição, com opções de visualizar, editar, cadastrar ou excluir dados institucionais;

- Gerenciar bibliotecários, podendo cadastrar, editar ou excluir, além de visualizar a lista atual.
- Após realizar qualquer uma dessas ações, o sistema fornece uma confirmação da operação.
- Além disso, o administrador pode gerar *backups* e visualizar relatórios, garantindo controle, segurança e rastreabilidade dos dados e das ações realizadas na instância da escola.

Esse diagrama destaca a função estratégica do administrador como gestor da estrutura da instituição, com foco em segurança, personalização e governança da plataforma BibliONtec.

3.2.3 – BPMN do aluno

Figura 39 - BPMN do Aluno



FONTE: AUTORIA PRÓPRIA (2025)

O diagrama BPMN do Aluno, representado na figura 39, demonstra de forma organizada as etapas que esse usuário percorre ao interagir com a plataforma para consultar, reservar e explorar o acervo de livros. O processo inicia-se com o acesso

ao sistema, no qual o aluno realiza o *login* e tem suas credenciais verificadas. Caso ocorra algum erro, como dados inválidos ou cadastro inexistente, o sistema retorna uma mensagem de aviso, impossibilitando o avanço. No entanto, quando a autenticação é bem-sucedida, o aluno passa a ter acesso às funcionalidades principais, como consultar notificações recebidas, visualizar recomendações de leitura personalizadas ou realizar pesquisas diretas no catálogo.

Ao efetuar uma busca, o aluno pode utilizar filtros e palavras-chave para localizar obras de interesse com maior precisão. Após escolher um livro específico, o sistema oferece diferentes possibilidades de interação:

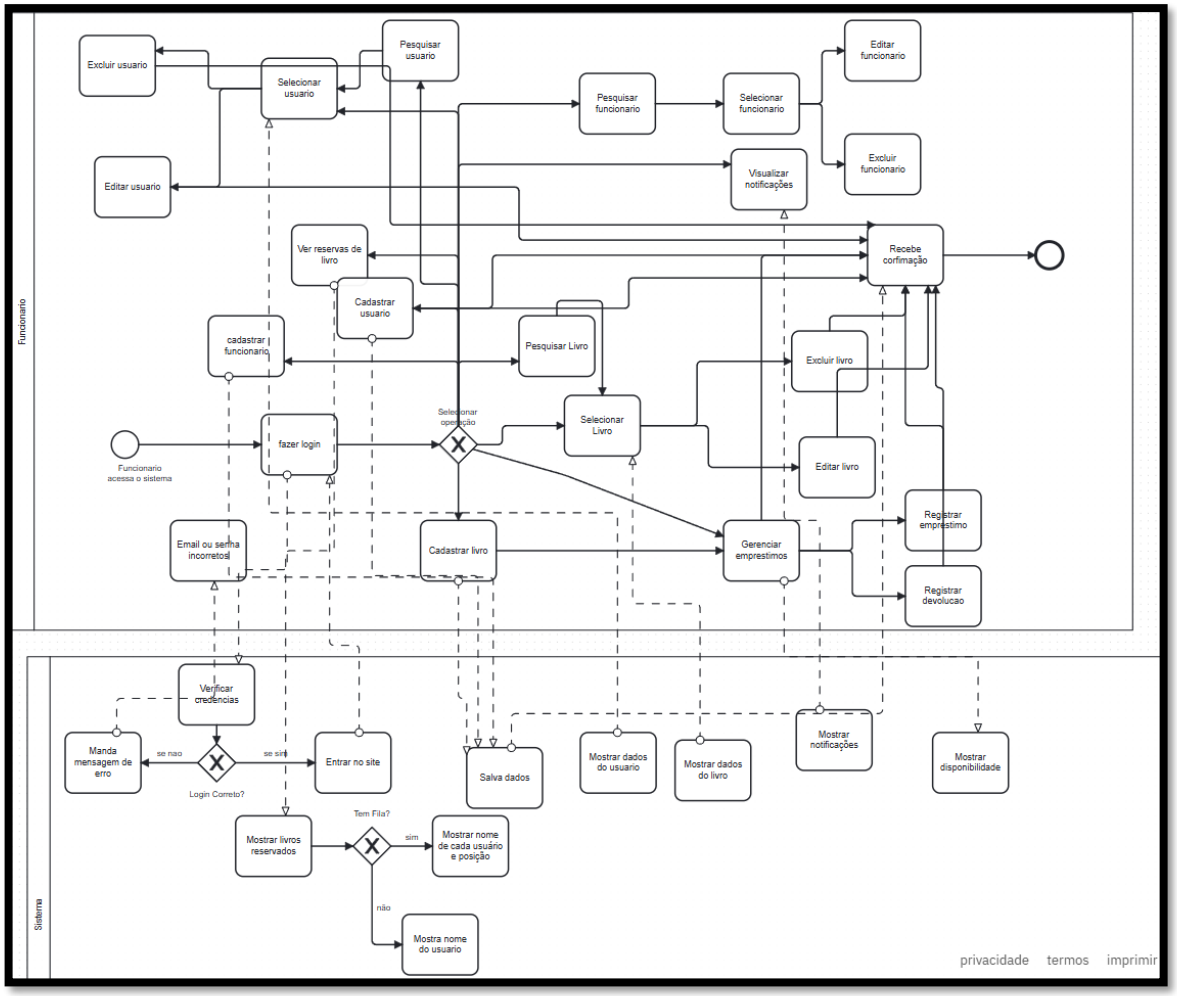
- Ver detalhes;
- Reservar (caso disponível);
- Adicionar à lista de desejo;
- Escrever comentários.

Essas opções ampliam a experiência de uso, permitindo que o aluno personalize sua relação com o acervo digital. Caso o título não esteja disponível no momento, a plataforma exibe essa informação de forma clara e possibilita que o aluno entre na fila de espera. Nesse cenário, o sistema organiza automaticamente a ordem dos solicitantes e envia notificações periódicas, tanto sobre a posição na fila quanto sobre a confirmação da reserva quando o item for liberado.

Além disso, o aluno pode solicitar a renovação do prazo de empréstimo, respeitando os limites estabelecidos, e será informado por meio de notificações automáticas sempre que atingir o número máximo permitido. Todo esse processo é apoiado por mecanismos de automação que garantem o envio de lembretes, a atualização contínua dos dados e o controle rigoroso das reservas. Dessa forma, o fluxo se torna mais confiável, transparente e eficiente, proporcionando ao aluno uma experiência prática e satisfatória no uso da plataforma BibliONtec.

3.2.4 – BPMN do funcionário (Bibliotecário)

Figura 40 - BPMN do Bibliotecário



FONTE: AUTORIA PRÓPRIA (2025)

O BPMN do Bibliotecário representado na figura 40 descreve as principais ações administrativas e operacionais que esse perfil executa na plataforma.

Após fazer *login*, o bibliotecário pode:

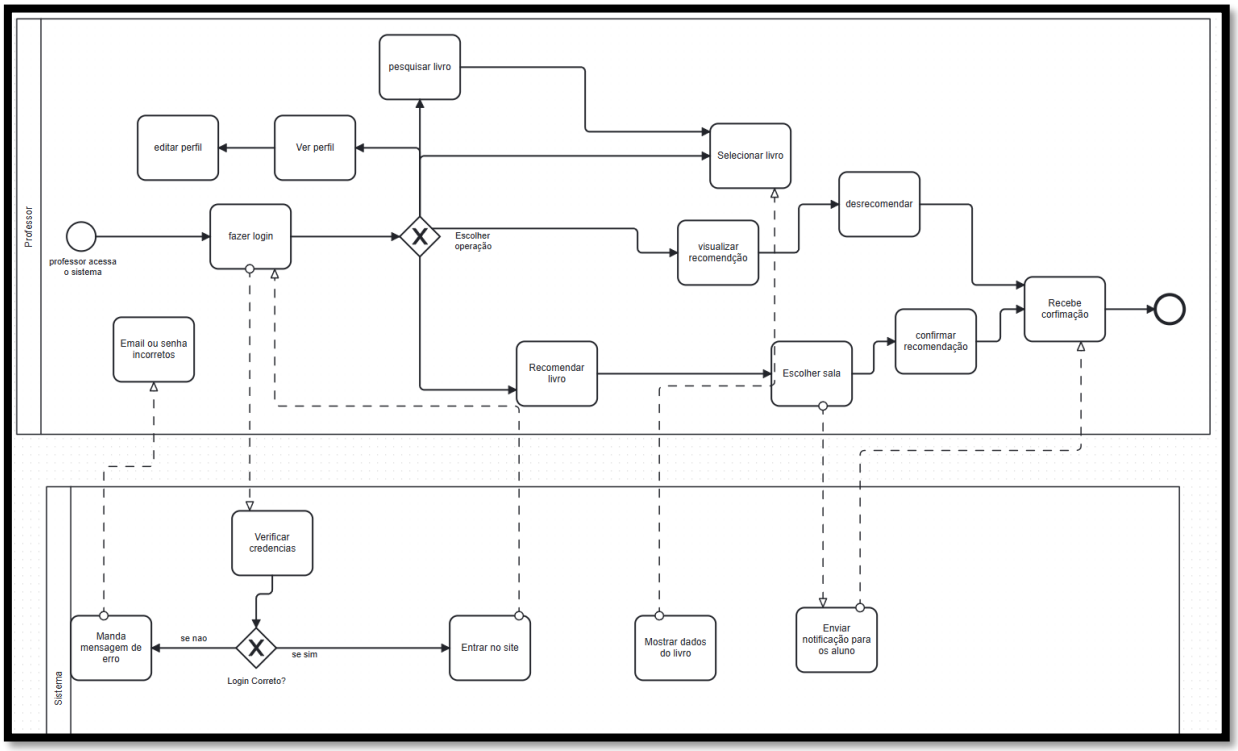
- Pesquisar, selecionar, cadastrar, editar ou excluir livros;
- Gerenciar empréstimos (registrar empréstimos e devoluções);
- Visualizar notificações;
- Cadastrar, visualizar, editar ou excluir usuários e funcionários;
- Ver reservas e fila de espera associadas aos livros.

Todas essas ações são conectadas ao sistema, que verifica credenciais, salva dados, exibe confirmações e mostra dados de usuários, livros e disponibilidade.

O bibliotecário é o principal responsável pela operação do acervo, controle dos usuários e manutenção da integridade das informações.

3.2.5 – BPMN do professor

Figura 41 - BPMN do Professor



FONTE: AUTORIA PRÓPRIA (2025)

O diagrama BPMN do Professor representado na figura 41 representa seu fluxo de uso no sistema, com foco em recomendar livros para alunos.

Após realizar *login* e ter o acesso validado, o professor pode:

- Ver e editar seu perfil;
- Pesquisar livros, selecionar e visualizar recomendações;
- Recomendar ou remover recomendações;
- Escolher a sala para a qual deseja recomendar o livro;
- Confirmar a recomendação;

O sistema, ao final, envia uma notificação automática para os alunos com a recomendação registrada. Todos os dados são validados e armazenados pelo sistema, que também exibe as informações do livro e confirma a ação realizada.

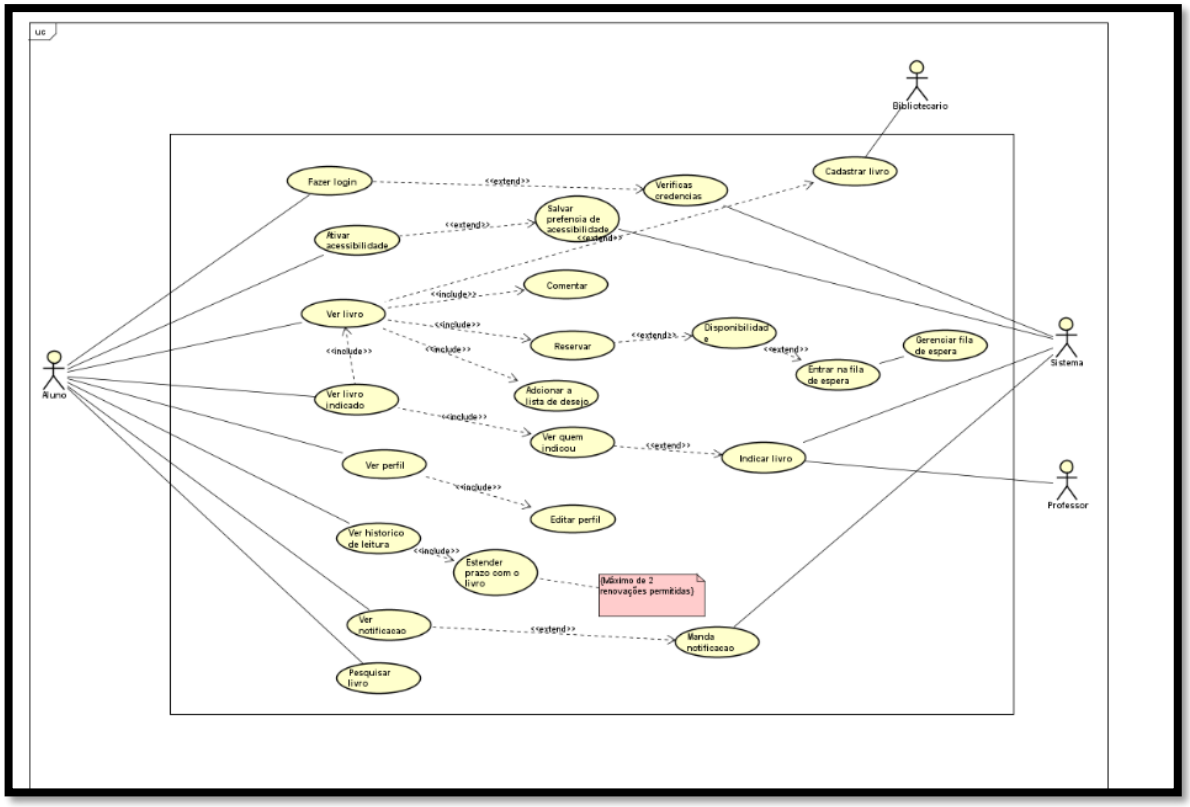
3.3 – DIAGRAMA

Um diagrama é uma representação gráfica simplificada que demonstra a aparência, estrutura ou funcionamento de algo, sendo amplamente utilizado tanto em ambientes acadêmicos quanto empresariais. Ele permite organizar visualmente grandes volumes de informações por meio de formas como retângulos, círculos, setas e linhas, o que facilita a compreensão e a tomada de decisões de maneira mais rápida e segura. A escolha do tipo de diagrama ideal depende do objetivo da análise. Por exemplo, o diagrama de árvore é usado para estruturar dados de maneira hierárquica, enquanto o diagrama de Venn mostra relações entre conjuntos. O diagrama de Pareto, por sua vez, é útil para identificar os problemas mais críticos de um processo, com base no princípio de que a maioria dos efeitos vem de poucas causas (MARIANO, 2025).

Os diagramas UML são direcionados ao desenvolvimento de *software*, pois representam estruturas e comportamentos de sistemas complexos, embora não sejam linguagens de programação. Já os mapas mentais são excelentes para organização de ideias e conteúdos, pois permitem a visualização geral de um tema por meio de ramificações que conectam informações. Atualmente, diversas ferramentas digitais facilitam a criação de diagramas. O Lucidchart é uma plataforma *online* que oferece modelos prontos e permite colaboração em tempo real. O Miro também é uma ferramenta colaborativa e intuitiva, ideal para o trabalho em equipe. O PowerPoint, embora mais tradicional, continua sendo uma opção eficiente e prática, principalmente para quem já domina a ferramenta. Além disso, a utilização de diagramas contribui para a redução de erros de interpretação, já que a informação passa a ser transmitida de forma mais objetiva e visual. Eles também estimulam a criatividade, tornando a análise de problemas mais dinâmica e favorecendo a geração de soluções inovadoras. Dessa forma, os diagramas se mostram essenciais para a análise, planejamento e comunicação clara de processos em diversas áreas do conhecimento (MARIANO, 2025).

3.3.1 – Diagrama do aluno

Figura 42 - Diagrama do Aluno



FONTE: AUTORIA PRÓPRIA (2025)

O diagrama apresentado na figura 42 representa as interações do Aluno com o sistema BibliONtec, destacando todas as funcionalidades disponíveis para este perfil. O ator “Aluno” está conectado a diversas ações que ele pode realizar diretamente no sistema, enquanto as setas e relacionamentos *include* (<<*include*>>) e *extend* (<<*extend*>>) mostram dependências ou complementações de funcionalidades.

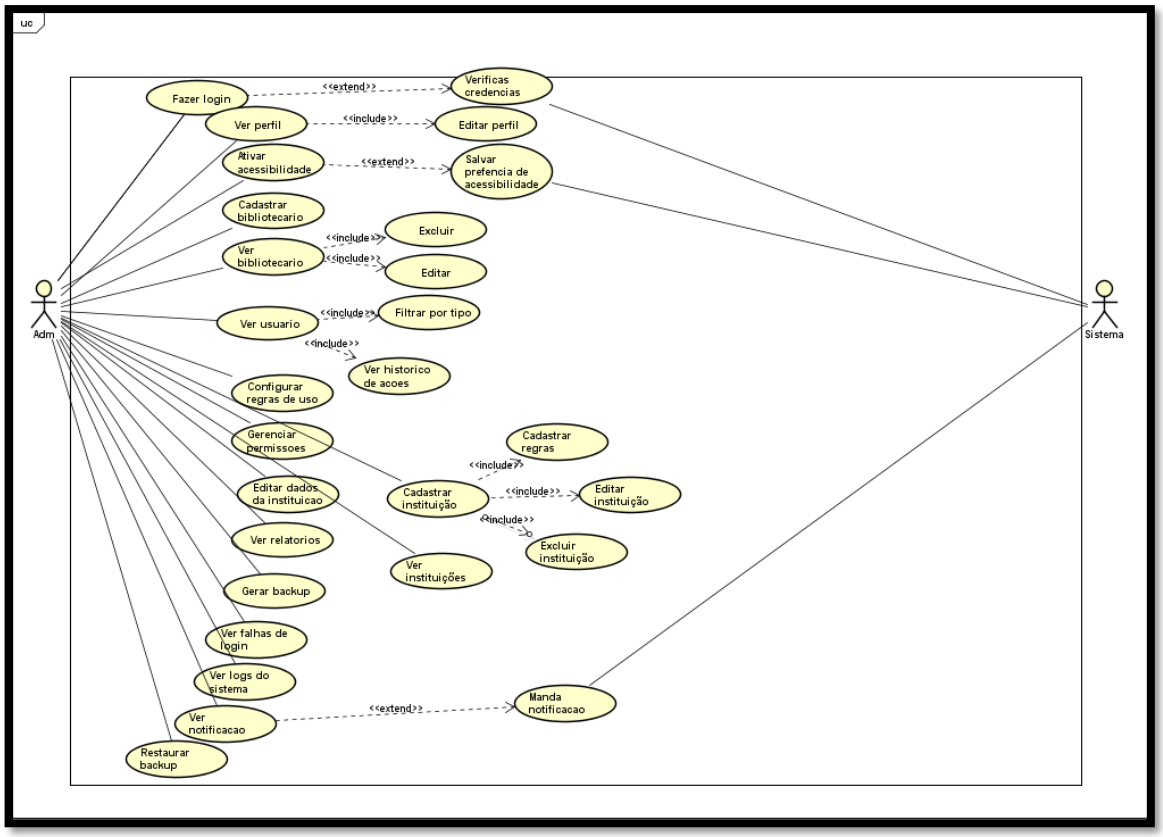
O acesso ao sistema se inicia pelo *login*, que aciona a verificação de credenciais e, opcionalmente, a funcionalidade de ativar acessibilidade, permitindo salvar preferências relacionadas a esse recurso. Entre as ações voltadas ao acervo, o Aluno pode visualizar livros, ver livros indicados e pesquisar obras específicas. Ao acessar um livro, é possível realizar interações adicionais, como comentar, reservar, ver disponibilidade, entrar na fila de espera, adicionar à lista de desejos ou ver quem indicou determinada obra.

O sistema também permite que o Aluno acompanhe seu histórico de leitura, incluindo a possibilidade de estender o prazo de empréstimo, com limite máximo de duas renovações. Outras funções contemplam o acesso e edição do perfil, bem como a visualização de notificações enviadas pelo sistema.

A interação do Aluno com bibliotecários e professores também é contemplada indiretamente, visto que essas funções podem cadastrar livros e gerenciar filas de espera, afetando as operações acessíveis ao aluno. Assim, este diagrama apresenta uma visão clara e detalhada do papel do Aluno no sistema BibliONtec, evidenciando como suas ações se integram ao fluxo geral de uso da plataforma.

3.3.2 – Diagrama do administrador

Figura 43 - Diagrama do administrador



FONTE: AUTORIA PRÓPRIA (2025)

O diagrama apresentado na figura 43 representa as interações do Administrador com o sistema BibliONtec, evidenciando todas as funcionalidades disponíveis para este perfil. Cada elipse indica um caso de uso, ou seja, uma ação ou funcionalidade que pode ser executada, enquanto as linhas conectando os atores aos

casos de uso mostram a relação direta entre o Administrador e as operações que ele pode realizar. Também estão presentes relacionamentos do tipo *include* (<<*include*>>) e *extend* (<<*extend*>>), que indicam dependências ou extensões de funcionalidades.

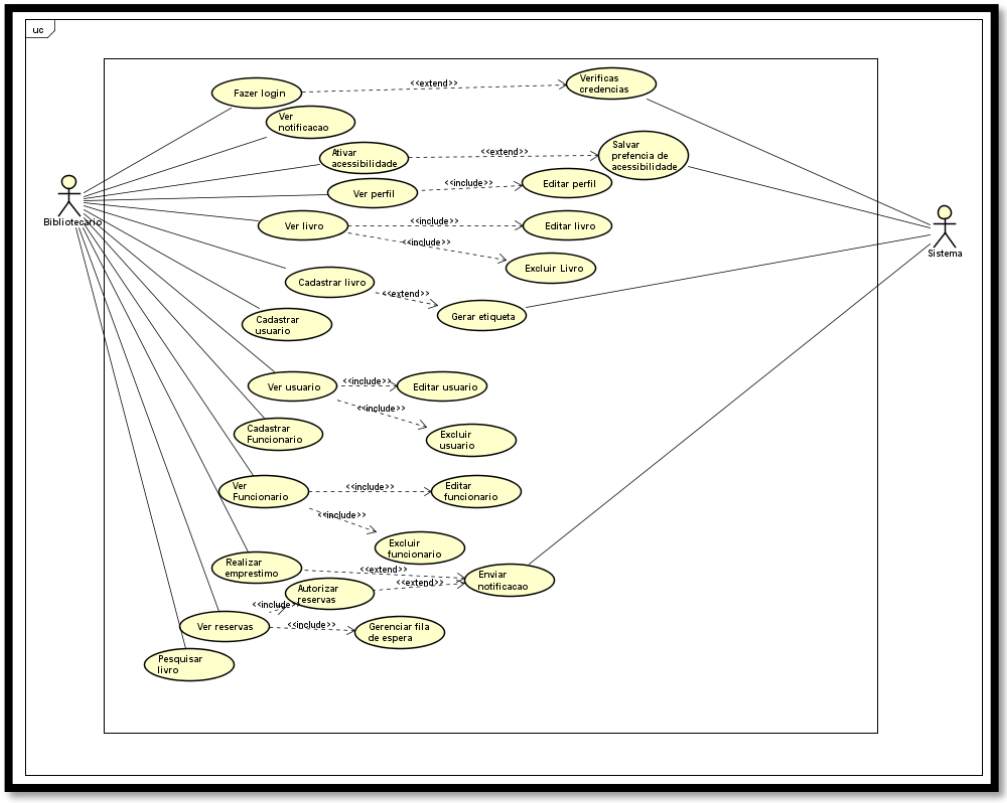
O fluxo de interação se inicia com o *login*, que inclui a verificação de credenciais pelo sistema. Uma vez autenticado, o Administrador pode acessar seu perfil, onde há possibilidade de editar informações e salvar preferências de acessibilidade. Ele também pode gerenciar bibliotecários e usuários, com opções de visualização, edição, exclusão e filtragem por tipo, além de consultar o histórico de ações realizadas no sistema.

No nível de gestão, o Administrador pode configurar regras de uso, gerenciar permissões e editar dados institucionais. Também é possível cadastrar novas instituições, editar ou excluir as existentes, bem como cadastrar e gerenciar regras associadas. A figura ainda evidencia que o Administrador pode visualizar relatórios, monitorar falhas de *login* e consultar *logs* do sistema.

Há também funcionalidades relacionadas à segurança e integridade dos dados, como gerar *backup* e restaurar *backup*. Outro ponto importante é o gerenciamento de notificações, que podem ser visualizadas pelo Administrador e enviadas pelo sistema. Além disso, observa-se que esse perfil desempenha papel estratégico, uma vez que suas ações impactam diretamente o funcionamento global da plataforma. A amplitude das permissões assegura que o Administrador tenha controle centralizado, mas também exige responsabilidade quanto ao uso consciente dessas ferramentas. Dessa maneira, o diagrama reforça não apenas as possibilidades de ação, mas também a importância desse usuário para a manutenção da confiabilidade, eficiência e segurança do BibliONtec. Em síntese, o Administrador atua como o elo entre a gestão técnica e a gestão operacional, garantindo que os recursos do sistema estejam alinhados às necessidades da instituição. Sua atuação é, portanto, indispensável para assegurar a evolução contínua e o bom desempenho da plataforma.

3.3.3 – Diagrama do bibliotecário

Figura 44 - Diagrama do Bibliotecário



FONTE: AUTORIA PRÓPRIA (2025)

O diagrama apresentado na figura 44 descreve as interações do Bibliotecário com o sistema BibliONtec, evidenciando as funcionalidades e permissões específicas atribuídas a este perfil. O ator “Bibliotecário” está conectado a múltiplos casos de uso, com relações de dependência (*include*) e extensão (*extend*) que mostram a interligação entre as ações.

O acesso ao sistema inicia-se com o *login*, que envolve a verificação de credenciais, podendo também acionar a funcionalidade de ativar acessibilidade, cujas preferências podem ser salvas para uso futuro. O Bibliotecário pode visualizar e editar seu perfil, ver notificações enviadas pelo sistema e realizar pesquisa de livros no acervo.

Entre as funções principais, destaca-se o cadastro de livros, que pode ser seguido pela edição ou exclusão de obras e pela geração de etiquetas para identificação física. O Bibliotecário também é responsável pelo cadastro e gerenciamento de usuários, podendo visualizar, editar ou excluir registros. Funções

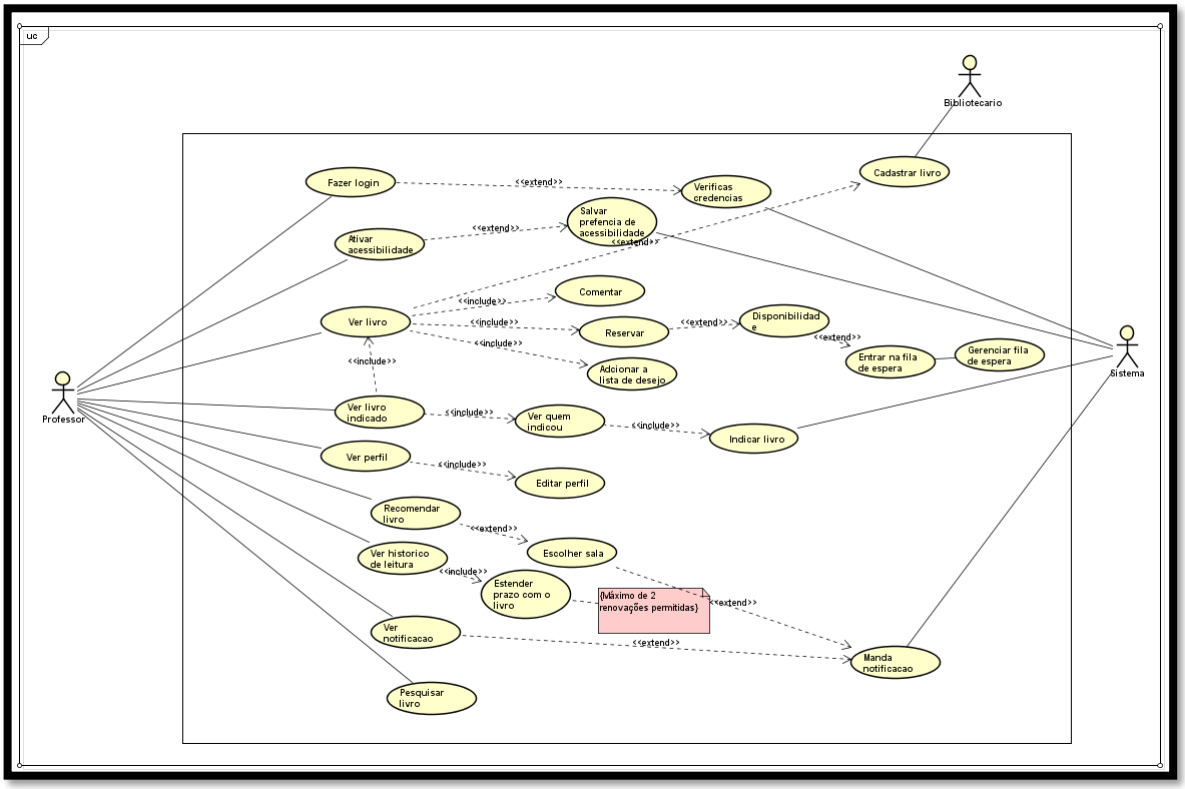
semelhantes se aplicam ao cadastro e gerenciamento de funcionários, com possibilidade de edição e exclusão desses registros.

No gerenciamento de empréstimos e reservas, o Bibliotecário pode realizar empréstimos, autorizar reservas e gerenciar a fila de espera de exemplares. Além disso, tem a capacidade de visualizar reservas ativas e enviar notificações aos usuários, de forma direta ou como resultado de outras operações.

Esse conjunto de funcionalidades posiciona o Bibliotecário como um dos principais mediadores do fluxo de informações e operações no BibliONtec, garantindo a organização do acervo, o atendimento aos usuários e a manutenção da base de dados do sistema.

3.3.4 – Diagrama do professor

Figura 45 - Diagrama do Professor



FONTE: AUTORIA PRÓPRIA (2025)

O diagrama de casos de uso apresentado na figura 45, referente ao professor, apresenta as principais interações desse ator com o sistema BibliONtec. Assim como o aluno, o professor tem acesso a funcionalidades básicas, como fazer *login*, ativar acessibilidade, visualizar livros, consultar notificações, editar perfil, pesquisar obras e

acompanhar seu histórico de leitura. Essas funções garantem a ele autonomia no gerenciamento de seu próprio uso do sistema.

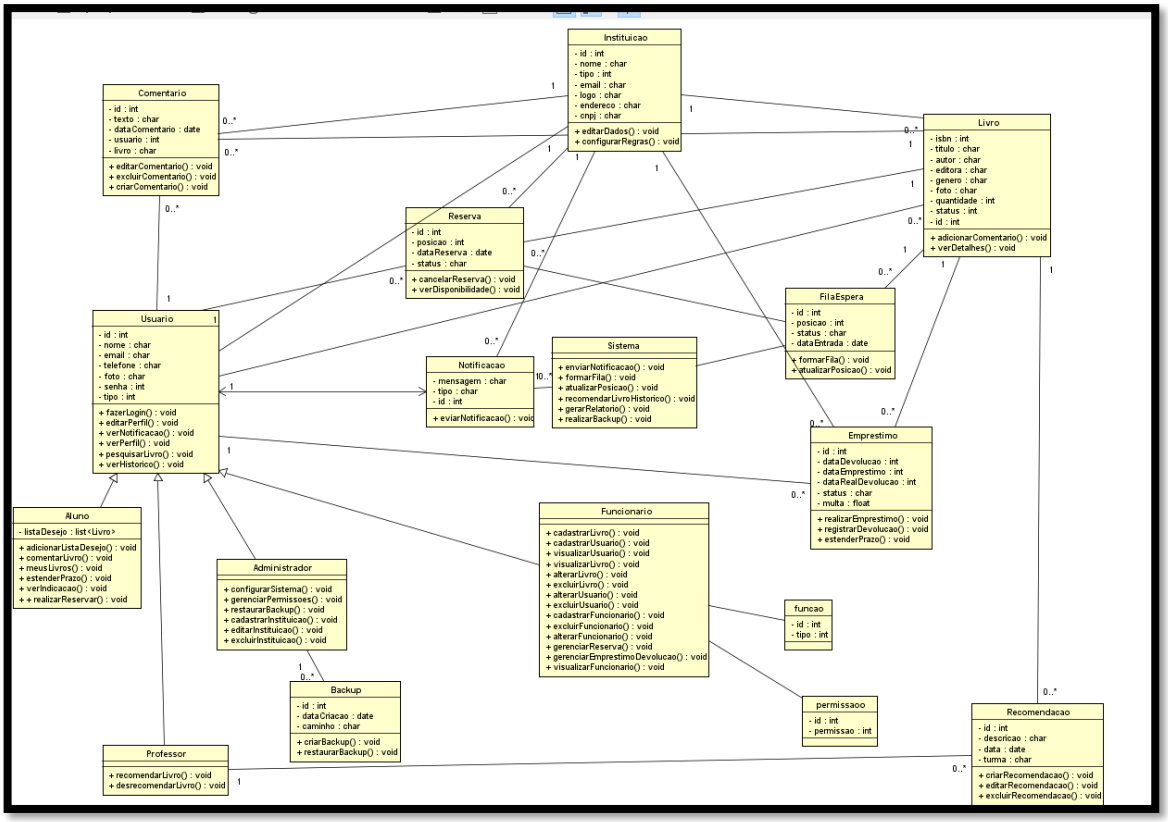
Entre as ações específicas desse ator, destaca-se a possibilidade de recomendar livros, funcionalidade que contribui para a curadoria e incentivo à leitura pelos estudantes. O professor pode indicar obras relevantes às suas turmas, tornando o sistema uma ferramenta pedagógica de apoio. Além disso, o professor possui a função de escolher sala, um caso de uso adicional em relação ao aluno, que reforça sua responsabilidade no contexto acadêmico e organizacional.

Assim como os outros usuários, o professor pode realizar reservas de livros, adicionar itens à lista de desejos, inserir comentários e consultar quem realizou recomendações. A gestão de prazos de empréstimos também está contemplada, com a possibilidade de solicitar a extensão do prazo de devolução, respeitando a regra estabelecida de no máximo duas renovações. O sistema, por sua vez, é responsável por enviar notificações e gerenciar a disponibilidade de exemplares, incluindo a organização da fila de espera quando necessário.

O diagrama evidencia, portanto, que o professor possui uma atuação híbrida dentro da aplicação: além de usufruir das funcionalidades comuns aos alunos, contribui ativamente para o enriquecimento da experiência da comunidade escolar por meio de recomendações e da gestão de ambientes de estudo. Essa modelagem ressalta a importância do professor não apenas como usuário, mas também como agente colaborativo no processo de ensino-aprendizagem apoiado pelo BibliONtec. Ademais, seu papel amplia a integração entre tecnologia e práticas pedagógicas, possibilitando maior dinamismo nas atividades escolares. A presença desse ator no sistema fortalece o vínculo entre a biblioteca digital e a sala de aula, transformando o BibliONtec em um recurso educacional de apoio contínuo. Dessa forma, percebe-se que o professor atua como mediador do conhecimento, incentivando o uso da leitura como ferramenta essencial para a formação crítica e intelectual dos estudantes.

3.3.5 – Diagrama de classe

Figura 46 - Diagrama de Classe



FONTE: AUTORIA PRÓPRIA (2025)

O diagrama de classes representado na figura 46 do sistema BibliONtec representa a estrutura fundamental da aplicação, organizando as entidades, seus atributos, métodos e os relacionamentos existentes entre elas. A classe Usuário ocupa posição central, possuindo atributos como id, nome, *e-mail*, telefone, senha e tipo, além de métodos voltados para o gerenciamento de perfil, notificações, histórico e pesquisas. A partir dela são especializadas outras classes, como Aluno, Professor, Administrador e Funcionário, que herdam as características principais de Usuário, mas adicionam comportamentos específicos conforme o papel desempenhado no sistema.

A classe Aluno contém métodos para adicionar livros à lista de desejos, comentar, visualizar histórico de leitura, estender prazos de empréstimos e realizar reservas. O Professor, por sua vez, possui funções relacionadas à recomendação de obras, enquanto o Administrador é responsável pela configuração do sistema, gerenciamento de instituições, permissões e processos de *backup*. Já o Funcionário

se concentra nas operações de cadastro, edição e exclusão de livros e usuários, bem como no gerenciamento de empréstimos e reservas, desempenhando um papel essencial no funcionamento cotidiano da biblioteca.

Outras classes complementam a modelagem e garantem a integridade dos processos. A classe Livro armazena informações como ISBN, título, autor, gênero e quantidade, permitindo adicionar comentários e visualizar detalhes. A classe Reserva registra dados de reservas realizadas, como posição, *status* e data, enquanto a classe Fila de Espera organiza os usuários interessados em exemplares indisponíveis. A classe Empréstimo mantém os dados referentes a retiradas de livros, com métodos para realizar, registrar devoluções e estender prazos. Também se destacam as classes Comentário e Recomendação, que viabilizam interações adicionais entre usuários e a biblioteca.

O sistema é apoiado ainda por classes de gerenciamento, como Notificação, responsável pelo envio de mensagens ao usuário, e *Backup*, que assegura a preservação dos dados. Além disso, a classe Instituição centraliza informações institucionais e permite a edição de dados e configuração de regras, enquanto as classes Função e Permissão garantem o controle de acesso e a correta hierarquização de privilégios dentro da aplicação.

Assim, o diagrama de classes evidencia a organização do sistema BibliONtec de maneira clara e estruturada, demonstrando como os diferentes elementos se relacionam para dar suporte às funcionalidades propostas. Essa modelagem assegura a coerência entre os requisitos levantados e a implementação do *software*, fornecendo uma visão abrangente e detalhada da arquitetura do sistema.

3.4 – APLICAÇÃO *WEB*

Com base na estrutura de classes definida, é possível visualizar como essas entidades ganham vida na prática por meio das funcionalidades oferecidas pela aplicação *web* do BibliONtec. O sistema a ser desenvolvido será uma aplicação *web* voltada para a gestão de uma biblioteca digital acadêmica. Os usuários poderão acessar a plataforma realizando *login* para consultar obras disponíveis, reservar livros, solicitar empréstimos e acompanhar notificações sobre prazos de devolução. A aplicação permitirá a navegação e pesquisa de títulos com detalhes como nome,

autor, capa e disponibilidade, possibilitando ao aluno reservar obras, adicioná-las à lista de desejos ou escrever comentários e avaliações. O processo de empréstimo será registrado pelo sistema, que também ficará responsável por enviar alertas automáticos sobre devoluções, penalidades por atraso e atualizações de reserva. Além disso, os usuários poderão acompanhar sua posição em filas de espera e estender prazos de empréstimo de acordo com as regras estabelecidas. Para os bibliotecários, a plataforma oferecerá ferramentas para o gerenciamento do acervo, controle de reservas, acompanhamento de usuários, assim garantindo um gerenciamento eficiente e uma experiência prática, intuitiva e acessível para todos os envolvidos. Para os administradores a plataforma oferecerá emissão de relatórios, cadastro de regras, estatísticas de utilização, configurar o *backup* automático, ver históricos de *backups* e ver pessoas que acessaram o sistema.

3.4.1 – Tela de *login*

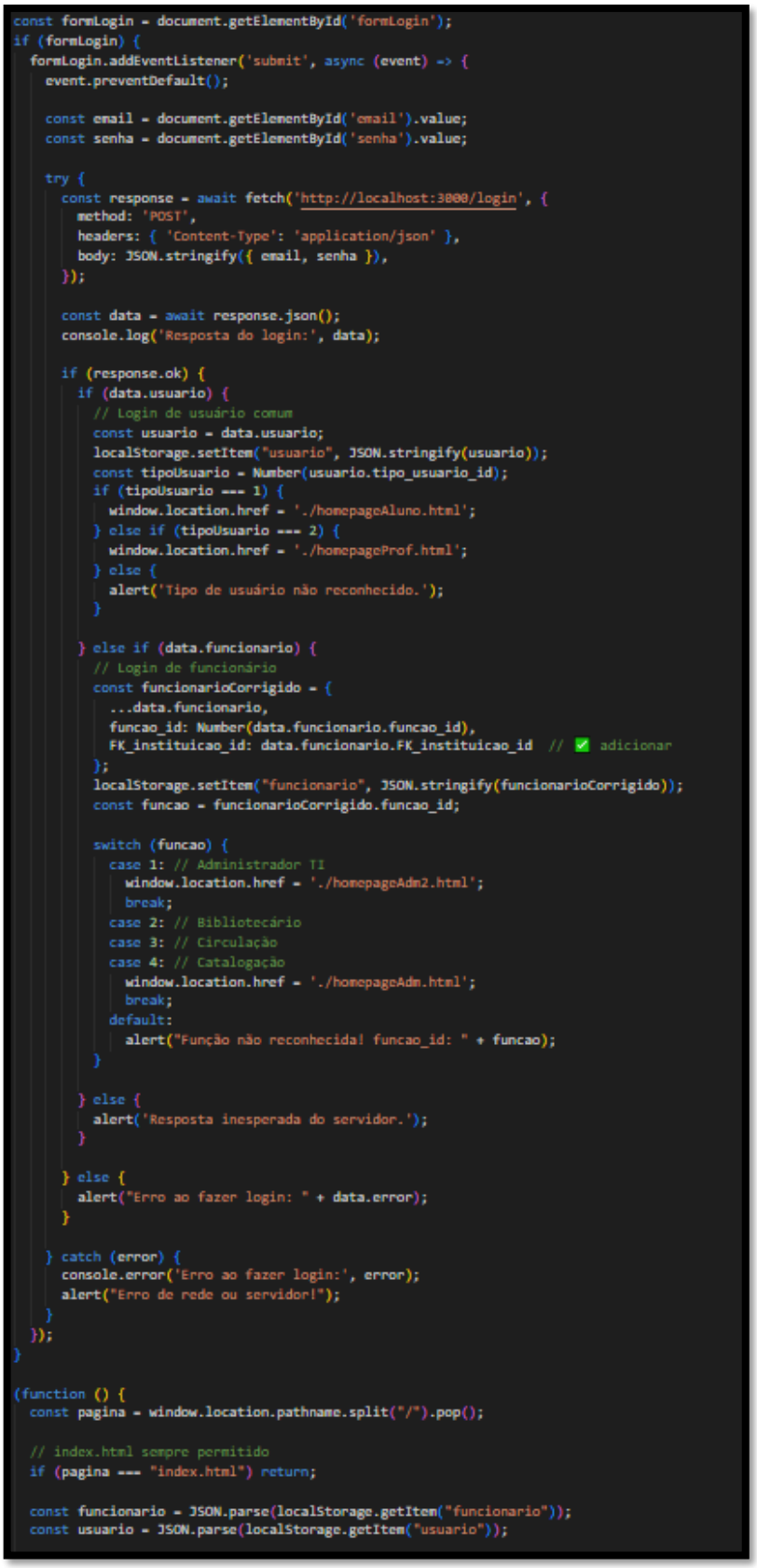
Figura 47 - Tela de *login*



FONTE: AUTORIA PRÓPRIA (2025)

A figura 47 mostra um formulário de *login*, onde o usuário precisa inserir seu *e-mail* e senha para acessar a aplicação. A senha está escondida por caracteres especiais que pode ser visualizada se clicar no ícone disponível, ela contém restrições a serem seguidas para o *login*. Também há uma opção para alterar a senha em caso de perda. O formulário contém um botão de confirmar para que o *login* seja realizado.

Figura 48 - Código de *Login*



```
const formLogin = document.getElementById('formLogin');
if (formLogin) {
  formLogin.addEventListener('submit', async (event) => {
    event.preventDefault();

    const email = document.getElementById('email').value;
    const senha = document.getElementById('senha').value;

    try {
      const response = await fetch('http://localhost:3000/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, senha }),
      });

      const data = await response.json();
      console.log('Resposta do login:', data);

      if (response.ok) {
        if (data.usuario) {
          // Login de usuário comum
          const usuario = data.usuario;
          localStorage.setItem("usuario", JSON.stringify(usuario));
          const tipoUsuario = Number(usuario.tipo_usuario_id);
          if (tipoUsuario === 1) {
            window.location.href = './homepageAluno.html';
          } else if (tipoUsuario === 2) {
            window.location.href = './homepageProf.html';
          } else {
            alert('Tipo de usuário não reconhecido.');
          }

        } else if (data.funcionario) {
          // Login de funcionário
          const funcionarioCorrigido = {
            ...data.funcionario,
            funcao_id: Number(data.funcionario.funcao_id),
            FK_instituicao_id: data.funcionario.FK_instituicao_id  // ✅ adicionar
          };
          localStorage.setItem("funcionario", JSON.stringify(funcionarioCorrigido));
          const funcao = funcionarioCorrigido.funcao_id;

          switch (funcao) {
            case 1: // Administrador TI
              window.location.href = './homepageAdm2.html';
              break;
            case 2: // Bibliotecário
            case 3: // Circulação
            case 4: // Catalogação
              window.location.href = './homepageAdm.html';
              break;
            default:
              alert("Função não reconhecida! funcao_id: " + funcao);
          }

        } else {
          alert('Resposta inesperada do servidor.');
        }

      } else {
        alert("Erro ao fazer login: " + data.error);
      }

    } catch (error) {
      console.error('Erro ao fazer login:', error);
      alert("Erro de rede ou servidor!");
    }
  });
}

(function () {
  const pagina = window.location.pathname.split("/").pop();

  // index.html sempre permitido
  if (pagina === "index.html") return;

  const funcionario = JSON.parse(localStorage.getItem("funcionario"));
  const usuario = JSON.parse(localStorage.getItem("usuario"));
```

FONTE: AUTORIA PRÓPRIA (2025)

A figura 48 mostra que este trecho é responsável pelo *login* de usuários comuns (alunos e professores) e funcionários. O formulário de *login* é capturado via *formLogin* e o evento *submit* é interceptado para que o envio seja feito de forma assíncrona com *fetch*. Os dados de *e-mail* e senha são enviados para a API (*login*) via método *POST*`. A resposta do servidor (*response*.json()) contém os dados do usuário em caso de sucesso.

Se o *login* for válido:

- Os dados do usuário são armazenados no *localStorage*.
- O sistema identifica o tipo de usuário através do campo tipo_usuario_id.
- Se tipo_usuario_id = 1, redireciona para *homepage*Aluno.html.
- Se tipo_usuario_id = 2, redireciona para *homepage*Prof.html.

Caso contrário, exibe uma mensagem de erro. Se a resposta não for válida, o sistema alerta sobre erro de *login* ou falha de rede. Esse código garante a autenticação de usuários comuns e os redireciona para a interface correta conforme seu perfil.

Logo em seguida mostra o processo de *login* para funcionários que após receber os dados do servidor (data.funcionario), é criado um objeto funcionarioCorrigido. Esse objeto contém todos os dados do funcionário, mas com o campo funcao_id convertido para número, evitando problemas de comparação. O funcionário corrigido é salvo no *localStorage*, permitindo que os dados fiquem disponíveis mesmo após mudanças de página.

Em seguida, o sistema utiliza um *switch* para redirecionar o usuário para a *homepage* adequada, de acordo com sua função:

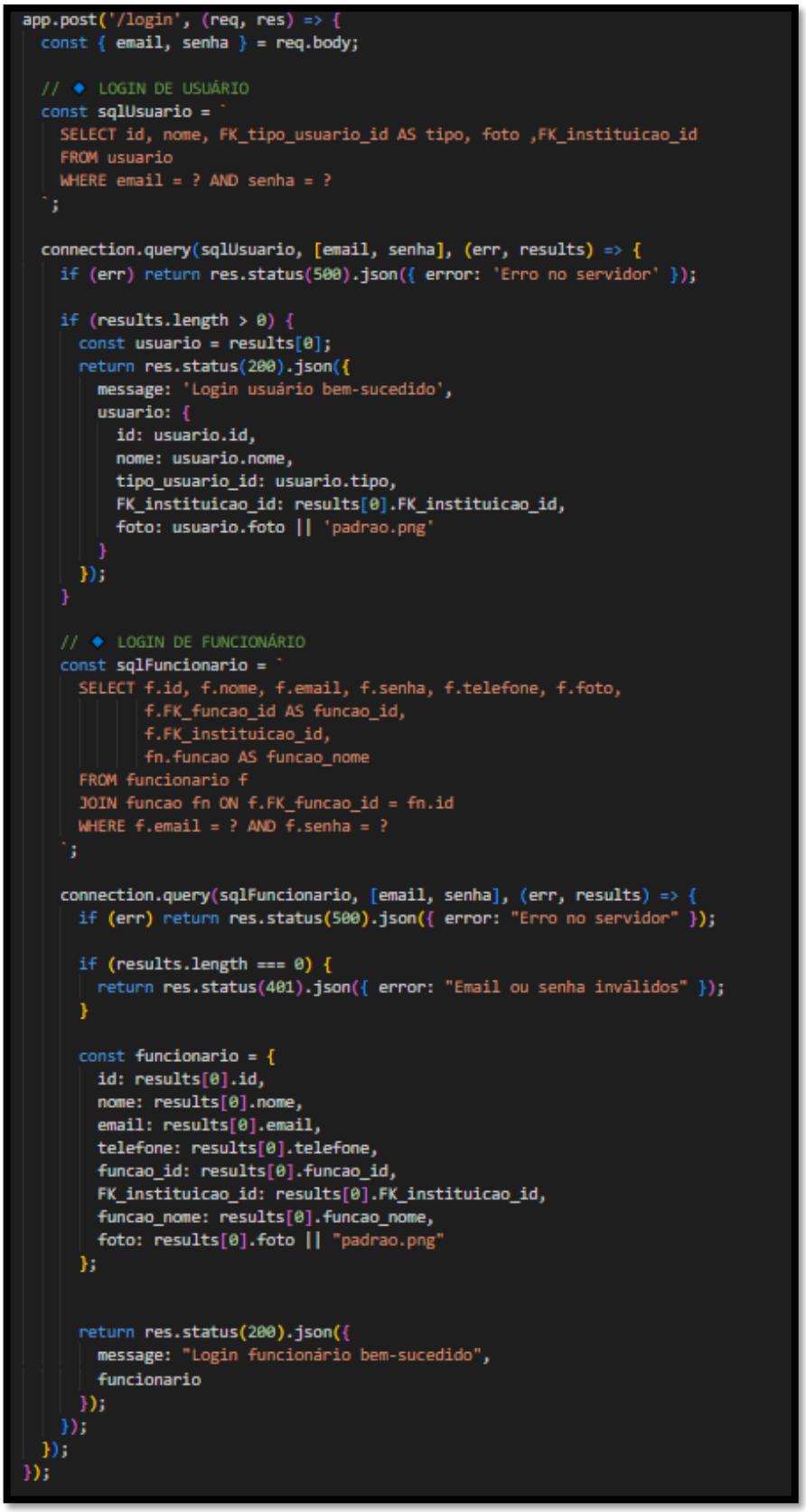
Função 1: Administrador de TI → *homepage*Adm2.html

Função 2, 3 ou 4: Bibliotecário, Circulação ou Catalogação → *homepage*Adm.html

Caso o sistema receba um funcao_id não reconhecido, exibe um alerta informando erro de função.

Esse código garante que cada funcionário seja direcionado para a página inicial compatível com sua função.

Figura 49 – Conexão com o banco de dados *login*



```
app.post('/login', (req, res) => {
  const { email, senha } = req.body;

  // 🔹 LOGIN DE USUÁRIO
  const sqlUsuario = `
    SELECT id, nome, FK_tipo_usuario_id AS tipo, foto ,FK_instituicao_id
    FROM usuario
    WHERE email = ? AND senha = ?
  `;

  connection.query(sqlUsuario, [email, senha], (err, results) => {
    if (err) return res.status(500).json({ error: 'Erro no servidor' });

    if (results.length > 0) {
      const usuario = results[0];
      return res.status(200).json({
        message: 'Login usuário bem-sucedido',
        usuario: {
          id: usuario.id,
          nome: usuario.nome,
          tipo_usuario_id: usuario.tipo,
          FK_instituicao_id: results[0].FK_instituicao_id,
          foto: usuario.foto || 'padrao.png'
        }
      });
    }

    // 🔹 LOGIN DE FUNCIONÁRIO
    const sqlFuncionario = `
      SELECT f.id, f.nome, f.email, f.senha, f.telefone, f.foto,
             f.FK_funcao_id AS funcao_id,
             f.FK_instituicao_id,
             fn.funcao AS funcao_nome
      FROM funcionario f
      JOIN funcao fn ON f.FK_funcao_id = fn.id
      WHERE f.email = ? AND f.senha = ?
    `;

    connection.query(sqlFuncionario, [email, senha], (err, results) => {
      if (err) return res.status(500).json({ error: "Erro no servidor" });

      if (results.length === 0) {
        return res.status(401).json({ error: "Email ou senha inválidos" });
      }

      const funcionario = {
        id: results[0].id,
        nome: results[0].nome,
        email: results[0].email,
        telefone: results[0].telefone,
        funcao_id: results[0].funcao_id,
        FK_instituicao_id: results[0].FK_instituicao_id,
        funcao_nome: results[0].funcao_nome,
        foto: results[0].foto || "padrao.png"
      };


      return res.status(200).json({
        message: "Login funcionário bem-sucedido",
        funcionario
      });
    });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

A figura 49 apresenta o trecho de código em Node.js responsável pela autenticação de usuários comuns e funcionários. O sistema utiliza a biblioteca *Express* para criar a rota / *login*, que recebe os dados de *e-mail* e senha enviados pelo formulário de *login*. Esses dados são transmitidos para o servidor através de uma requisição do tipo *POST*.

O *back-end* realiza a conexão com o banco de dados MySQL e executa uma consulta SQL para verificar a existência do usuário. Primeiramente, é realizada a verificação na tabela usuário, retornando informações como id, nome, tipo_usuario_id, FK_instituicao_id e foto. Se o *login* for válido:

- Os dados retornados são enviados em formato JSON para o front-end.
- O sistema identifica o tipo de usuário por meio do campo tipo_usuario_id.
- Se tipo_usuario_id = 1, o redirecionamento ocorre para homepageAluno.html.
- Se tipo_usuario_id = 2, o redirecionamento ocorre para homepageProf.html.

Caso o usuário não seja encontrado, o sistema segue para a próxima verificação.

Na ausência de correspondência na tabela de usuários, o sistema executa uma nova consulta SQL na tabela funcionário, unindo-a à tabela função para obter informações adicionais. O servidor retorna campos como id, nome, *e-mail*, telefone, funcao_id, FK_instituicao_id, funcao_nome e foto. Em seguida, é criado um objeto funcionarioCorrigido, no qual o campo funcao_id é convertido para número, prevenindo erros de comparação. Esse objeto é então armazenado no *localStorage*, permitindo que os dados permaneçam disponíveis durante a navegação.

O redirecionamento do funcionário ocorre através de uma estrutura *switch*, conforme o valor de funcao_id:

Função 1: Administrador de TI → *homepage*Adm2.html.

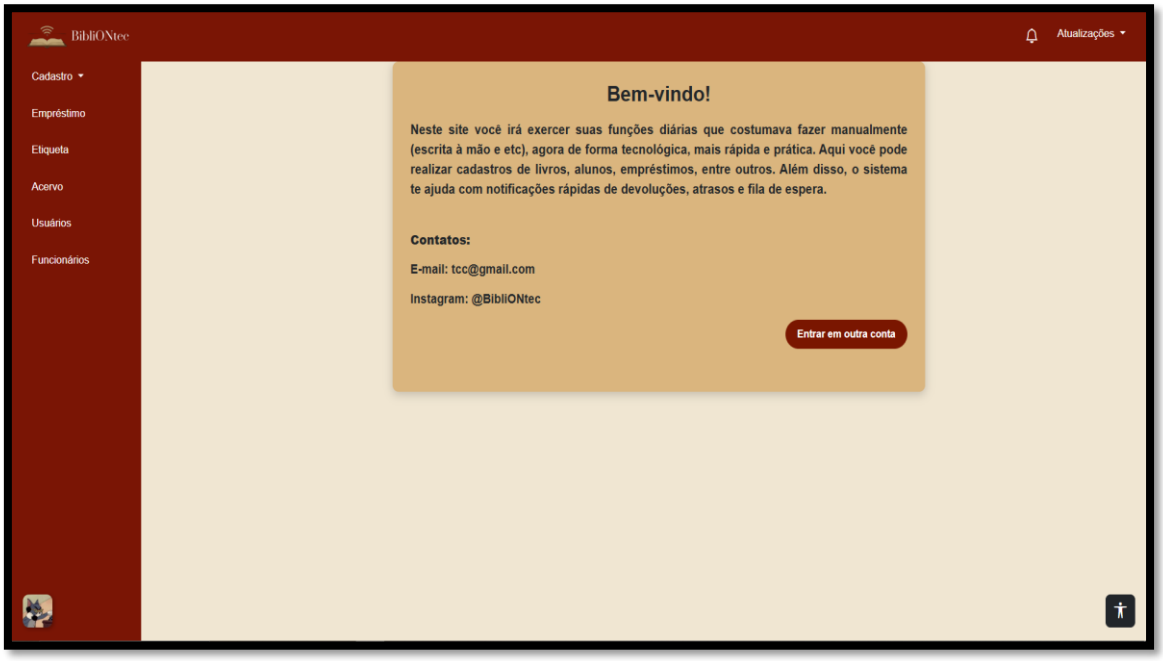
Funções 2, 3 ou 4: Bibliotecário, Circulação ou Catalogação → *homepage*Adm.html.

Caso seja identificado um funcao_id não reconhecido, o sistema emite um alerta de erro de função.

Esse processo garante que tanto usuários comuns quanto funcionários sejam autenticados de forma segura e direcionados automaticamente para a interface correspondente ao seu perfil ou função dentro da instituição.

3.4.2 – Tela Inicial do bibliotecário

Figura 50 - Tela Inicial bibliotecário



FONTE: AUTORIA PRÓPRIA (2025)

A figura 50 apresenta a tela inicial do sistema BibliONtec do bibliotecário, que tem como objetivo dar boas-vindas a ele e oferecer uma visão geral das principais funcionalidades.

Logo e nome do sistema (canto superior esquerdo): o logotipo da biblioteca, acompanhado do nome BibliONtec, identifica o sistema e garante a identidade visual do projeto.

Menu lateral (lado esquerdo, em vermelho)

Área central de boas-vindas: no centro da tela, há uma mensagem de “Bem-vindo!” que explica brevemente o propósito do sistema. O texto destaca que as funções da biblioteca antes feitas manualmente agora são realizadas de forma

tecnológica, ágil e prática, incluindo cadastros, empréstimos e notificações de devoluções e filas de espera.

Informações de contato: abaixo da mensagem principal, são apresentados os canais de contato do sistema:

E-mail: (tcc@gmail.com) para suporte ou dúvidas.

Instagram: (@BibliONtec), para comunicação e interação com os usuários em redes sociais.

Botão “Entrar em outra conta”: localizado no canto inferior do quadro central, esse botão permite que o usuário alterne o *login* e acesse o sistema com outro perfil.

Notificações (canto superior direito): há um ícone de sino com a palavra “Atualizações”, indicando que o usuário pode acessar novidades, avisos ou mensagens emitidas pelo sistema, uma funcionalidade importante no contexto dos usuários.

Rodapé e acessibilidade: no canto inferior direito, há um ícone de acessibilidade que, ao ser acessado, permite diferentes visualizações da aplicação ajustadas a determinados tipos de daltonismo, reforçando a preocupação do sistema com a inclusão de diferentes perfis de usuários. No canto inferior esquerdo, encontra-se um ícone de perfil, sugerindo que o usuário está logado em uma conta específica.

Além de transmitir uma identidade visual consistente, essa tela inicial também funciona como ponto de partida para todas as operações do bibliotecário, centralizando atalhos importantes em uma interface clara e bem organizada. A presença de ícones intuitivos facilita a aprendizagem de novos usuários e diminui a curva de adaptação. Outro ponto relevante é que, ao integrar informações institucionais e canais de contato já na tela inicial, o sistema fortalece a comunicação e aproxima o bibliotecário do suporte, garantindo maior confiabilidade no uso da plataforma.

Adicionalmente, essa tela inicial também transmite profissionalismo e organização, reforçando o papel do BibliONtec como uma solução moderna para a gestão de bibliotecas. A simplicidade de navegação somada à clareza visual facilita o trabalho cotidiano do bibliotecário e cria um ambiente mais acolhedor para quem

utiliza o sistema. Por fim, a estrutura apresentada serve como base para que novas funções sejam adicionadas futuramente sem comprometer a experiência do usuário, mantendo o equilíbrio entre eficiência e acessibilidade.

Figura 51 - Código HTML da Tela Inicial

```
<!-- Conteúdo -->
<div class="main-container">
  <div class="content">
    <h1>Bem-vindo!</h1>
    <p style="font-size: 20px; text-align: justify;">
      Neste site você irá exercer suas funções diárias que costumava fazer manualmente (escrita à mão e etc), agora
      de forma tecnológica, mais rápida e prática. Aqui você pode realizar cadastros de livros, alunos, empréstimos,
      entre outros. Além disso, o sistema te ajuda com notificações rápidas de devoluções, atrasos e fila de espera.
    </p>
    <br>
    <p><strong>Contatos:</strong></p>
    <p>E-mail: tcc@gmail.com</p>
    <p>Instagram: @BibliONtec</p>
    <div class="text-end">
      <a href="./index.html" class="btn btn-cadastro">Entrar em outra conta</a>
    </div>
  </div>
</div>
```

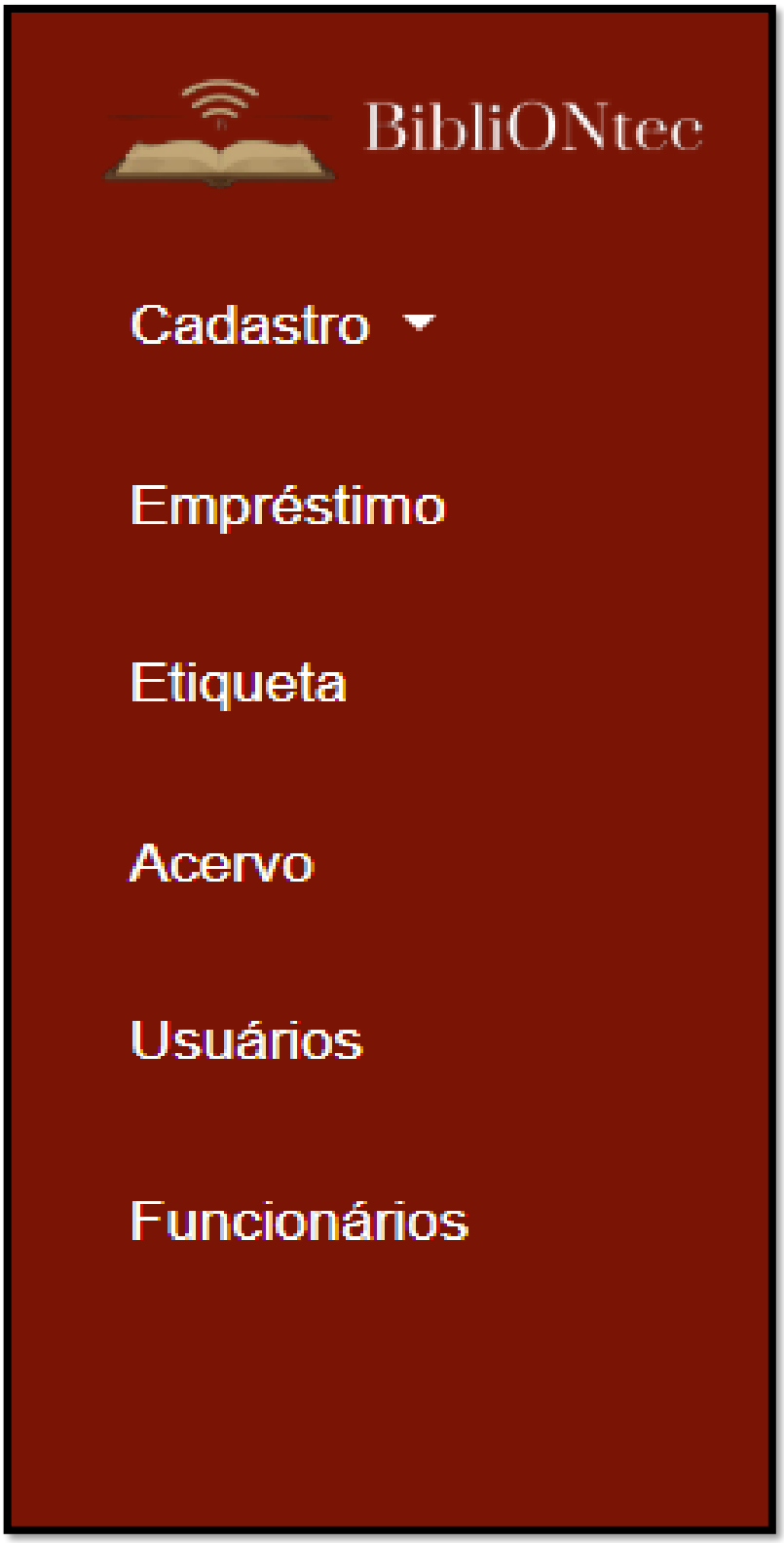
FONTE: AUTORIA PRÓPRIA (2025)

O código apresentado na figura 51 é responsável pela estruturação da página inicial do sistema BibliONtec, sendo desenvolvido na linguagem de marcação HTML. Essa estrutura é composta por um contêiner principal *main-container*, que centraliza o conteúdo exibido, e uma divisão interna *content*, responsável por organizar os elementos textuais e visuais.

Na parte superior, é utilizado um título principal h1 com a mensagem de boas-vindas ao usuário. Em seguida, há um parágrafo formatado com estilo CSS embutido, no qual se apresenta a descrição do sistema, destacando suas funcionalidades, como o cadastro de livros, alunos, empréstimos e o envio de notificações sobre devoluções, atrasos e fila de espera. Logo abaixo, são exibidas as informações de contato, incluindo *e-mail* e Instagram, para suporte e comunicação com os desenvolvedores.

Por fim, o código contém um *link* estruturado como botão, configurado para redirecionar o usuário à página de *login index*.html, permitindo o acesso ou troca de conta. Esse elemento está envolvido em uma divisão *text-end* que auxilia no posicionamento do botão dentro da página. Assim, a construção do código garante clareza, organização e usabilidade na tela inicial do sistema.

Figura 52 - Menu lateral do sistema



FONTE: AUTORIA PRÓPRIA (2025)

A figura 52 apresenta a interface gráfica do menu lateral do sistema BibliONtec, que é voltado para a gestão de bibliotecas. O fundo do menu possui um

tom vermelho escuro, conferindo um visual sóbrio e elegante à aplicação. O menu é composto por uma barra vertical posicionada à esquerda da tela, contendo os seguintes elementos:

Logotipo do Sistema: Na parte superior do menu, há um ícone representando um livro aberto com um símbolo de sinal de *Wi-Fi* acima, sugerindo a integração entre conhecimento (livro) e tecnologia (*Wi-Fi*). Ao lado do ícone, está o nome do sistema: BibliONtec, com destaque para a parte "*ON*", remetendo à ideia de tecnologia e conectividade.

Item de Menu – Cadastro: Logo abaixo do logotipo, encontra-se o item "Cadastro", acompanhado de um pequeno ícone de seta apontando para baixo, indicando que se trata de um menu suspenso com subitens (não visíveis nesta imagem).

Item de Menu – Empréstimo: Em seguida, há o item "Empréstimo", leva o usuário a funcionalidades relacionadas ao empréstimo de livros e materiais do acervo.

Item de Menu – Etiqueta: O item "Etiqueta" sugere uma funcionalidade voltada para a geração ou impressão de etiquetas, para catalogação de obras físicas.

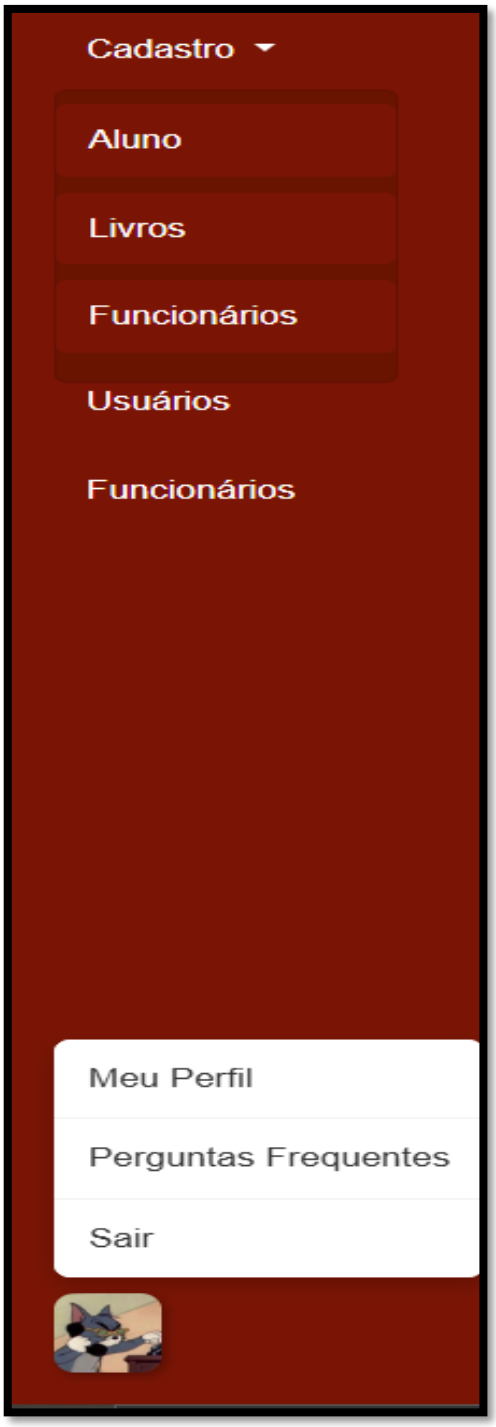
Item de Menu – Acervo: Este item permite o acesso à gestão do acervo da biblioteca, onde podem ser visualizados, cadastrados ou editados os livros e materiais disponíveis.

Item de Menu – Usuários: Refere-se à área de controle dos usuários cadastrados no sistema, como alunos, professores ou outros frequentadores da biblioteca.

Item de Menu – Funcionários: Indica uma seção voltada à administração dos funcionários que operam o sistema ou trabalham na biblioteca.

Cada item do menu está disposto de forma vertical, com texto branco para garantir legibilidade sobre o fundo escuro. Essa organização proporciona uma navegação intuitiva e eficiente para os usuários do sistema.

Figura 53 - Menu lateral expandido



FONTE: AUTORIA PRÓPRIA (2025)

A figura 53 exibe o menu lateral do sistema BibliONtec, com alguns elementos interativos expandidos. O menu continua mantendo o mesmo padrão visual da imagem anterior, reforçando a identidade visual do sistema e oferecendo uma navegação clara e intuitiva. A seguir, são descritos os principais elementos presentes na tela, que destacam as opções interativas de forma mais clara para o usuário.

Menu “Cadastro” Expandido: O item "Cadastro" aparece com seu submenu expandido, exibindo três opções de cadastro distintas e funcionais. São elas:

Aluno: Direciona para a tela específica de cadastro de alunos, permitindo que o usuário adicione novos estudantes ao sistema.

Livros: Oferece acesso direto à tela de cadastro de obras do acervo, onde os livros podem ser registrados ou atualizados no sistema de gestão da biblioteca.

Funcionários: Opção destinada ao cadastro de colaboradores ou funcionários que atuam na biblioteca, facilitando o gerenciamento dos recursos humanos da instituição.

Outros Itens de Menu: Abaixo do menu "Cadastro", estão listados itens adicionais que proporcionam acesso a outras funcionalidades do sistema, garantindo que o usuário encontre rapidamente os módulos de interesse. Essa estrutura de menu favorece a organização e contribui para a eficiência do uso diário da ferramenta.

Menu de Perfil do Usuário (Inferior Direito): No canto inferior esquerdo da tela, aparece uma foto de perfil em miniatura, que indica a área pessoal do usuário logado no sistema. Ao clicar nesta imagem, é exibido um pequeno menu com três opções rápidas e práticas para o usuário:

Meu Perfil: Redireciona o usuário para a área onde ele pode editar ou visualizar os próprios dados cadastrados no sistema, como nome, *e-mail* e configurações de conta.

Perguntas Frequentes: Leva o usuário a uma seção de ajuda, o que pode ser útil em caso de problemas ou incertezas no uso do sistema.

Sair: Um botão claro e visível para realizar o *logout* do sistema, garantindo a segurança e a privacidade do usuário ao terminar sua sessão.

Esse conjunto de elementos torna a navegação mais acessível, reforçando a usabilidade da aplicação. Além disso, a disposição intuitiva do menu contribui para que até novos usuários se adaptem rapidamente ao ambiente do BibliONtec.

Figura 54 - Código do menu lateral

```
<div class="sidebar">
  <div class="dropdown" data-permissao="cadastro">
    <button class="btn dropdown-toggle w-100" type="button" data-bs-toggle="dropdown">Cadastro</button>
    <ul class="dropdown-menu">
      <li><a class="dropdown-item" href="./cadAluno.html" data-permissao="cadAluno">Aluno</a></li>
      <li><a class="dropdown-item" href="./cadLivro.html" data-permissao="cadLivro">Livro</a></li>
      <li><a class="dropdown-item" href="./cadFunc.html" data-permissao="cadFunc">Funcionário</a></li>
    </ul>
  </div>
  <a class="btn w-100" href="./emprestimo.html" data-permissao="emprestimo">Empréstimo</a>
  <a class="btn w-100" href="./etiqueta.html" data-permissao="etiqueta">Etiqueta</a>
  <a class="btn w-100" href="./areaAdm.html" data-permissao="configAdm2">Administrador</a>
  <a class="btn w-100" href="./acervo.html" data-permissao="acervo">Acervo</a>
  <a class="btn w-100" href="./usuarios.html" data-permissao="usuarios">Usuários</a>
  <a class="btn w-100" href="./funcionarios.html" data-permissao="funcionarios">Funcionários</a>
</div>
<!-- Avatar com menu fixado no canto inferior esquerdo -->
<div class="avatar-container">
  <img src="http://localhost:3000/uploads/padrao.jpg" id="avatarPerfil" class="avatar-navbar" alt="Avatar do Usuário" />
  <div id="menuPerfil" class="menu-perfil">
    <a href="./areaAdm.html" class="dropdown-item">Meu Perfil</a>
    <hr class="menu-divider">
    <a href="#" class="dropdown-item">Perguntas Frequentes</a>
    <hr class="menu-divider">
    <a href="./index.html" class="dropdown-item" onclick="logout()">Sair</a>
  </div>
</div>
```

FONTE: AUTORIA PRÓPRIA (2025)

O código apresentado na figura 54 corresponde à estrutura da barra lateral de navegação do sistema, responsável por organizar as principais funcionalidades do *software*.

Na parte superior, encontra-se o menu Cadastro, implementado no formato de um *dropdown*. Esse menu possibilita o acesso a três tipos de cadastro distintos: aluno, livro e funcionário. Cada item redireciona o usuário para a respectiva página de registro, sendo controlado por um sistema de permissões que garante que apenas usuários autorizados possam visualizar ou acessar essas opções.

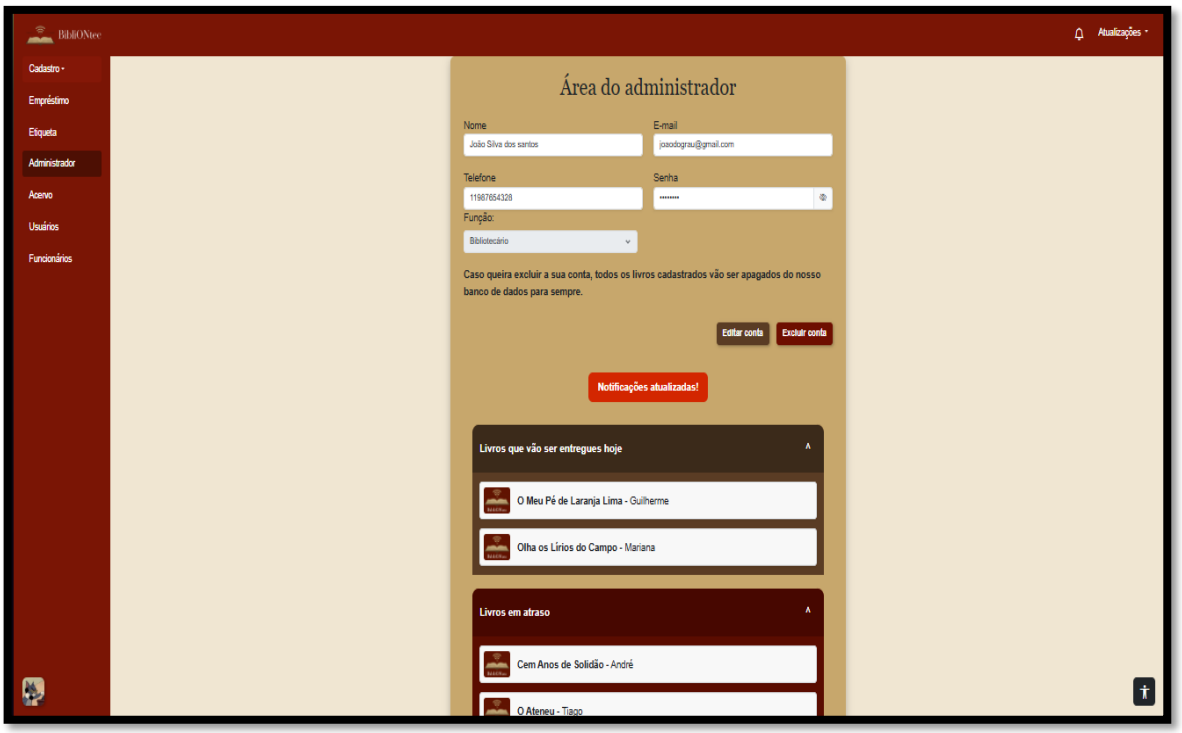
Logo abaixo, a barra lateral contém botões de acesso direto para outras funcionalidades centrais do sistema, como: empréstimo, etiqueta, administrador, acervo, usuários e funcionários. Assim como no menu de cadastro, cada um desses botões é vinculado a permissões específicas, assegurando a gestão hierárquica do uso da aplicação.

Na parte inferior, há um menu de perfil do usuário, associado uma miniatura da foto de perfil fixada no canto esquerdo da tela. Esse menu oferece três opções: Meu Perfil, que direciona para a área administrativa do usuário; Perguntas Frequentes, destinada ao suporte rápido; e a opção Sair, que executa a função de *logout* do sistema.

Dessa forma, a organização da interface visa tornar a navegação prática e intuitiva, ao mesmo tempo em que mantém um controle de acesso eficiente por meio do uso de permissões.

3.4.3 – Área do bibliotecário

Figura 55 - Tela área bibliotecário



FONTE: AUTORIA PRÓPRIA (2025)

A tela apresentada na figura 55 corresponde à Área do Bibliotecário, uma das principais interfaces do sistema. Ela foi desenvolvida para oferecer ao bibliotecário um espaço centralizado, no qual ele pode visualizar e gerenciar suas informações pessoais, além de acompanhar atividades relacionadas à biblioteca.

Na parte superior da tela, há um formulário contendo campos como Nome, *E-mail*, Telefone, Senha e Função. Esses dados são carregados automaticamente do banco de dados assim que o bibliotecário acessa sua conta, garantindo que ele possa

visualizar suas informações já registradas. Além disso, o sistema permite que o bibliotecário edite seus dados sempre que necessário, ou até mesmo exclua sua conta permanentemente. Ao excluir a conta, há um aviso claro de que os livros cadastrados por esse usuário também serão apagados do banco de dados, reforçando a importância dessa ação.

Logo abaixo, há um botão de notificações atualizadas, que funciona como um recurso de alerta rápido, permitindo que o bibliotecário se mantenha informado sobre os principais eventos da biblioteca, como devoluções, atrasos e entregas programadas.

Na parte inferior, a tela traz dois painéis de grande relevância para a rotina do bibliotecário:

- Livros que vão ser entregues hoje: lista de títulos que estão previstos para devolução no dia, já vinculados ao respectivo usuário que fez o empréstimo. Esse recurso facilita o controle e a organização das devoluções, ajudando o bibliotecário a se preparar com antecedência.
- Livros em atraso: exibe os títulos que não foram devolvidos dentro do prazo estipulado, também identificando os usuários responsáveis. Essa funcionalidade permite que o bibliotecário acompanhe de forma clara os casos de atraso e, se necessário, entre em contato com os usuários para regularização.

Dessa forma, a Área do Bibliotecário não é apenas um espaço de gestão de perfil, mas também um painel de controle completo, reunindo informações pessoais, notificações importantes e dados operacionais relacionados à biblioteca, tornando o trabalho mais organizado e eficiente.

Além disso, essa tela reflete a integração entre o *front-end* e o *back-end* do sistema, já que cada ação realizada pelo bibliotecário, como atualização de dados ou acompanhamento de empréstimos, é processada de forma dinâmica e segura no banco de dados. Essa comunicação garante maior confiabilidade das informações apresentadas, evitando inconsistências e contribuindo para que o bibliotecário tenha sempre uma visão atualizada do funcionamento da biblioteca.

Figura 56 - Código para carregamento de usuário A

```
//============================== Função genérica para carregar dados e preencher campos ================================

async function carregarDados(id, tipo) {
  try {
    const endpoint = tipo === "funcionario" ? "funcionarios" : "usuario";
    const token = localStorage.getItem("token");

    const res = await fetch(`http://localhost:3000/api/${endpoint}/${id}`, {
      headers: { "Authorization": "Bearer " + token }
    });

    if (!res.ok) throw new Error(`Erro HTTP! status: ${res.status}`);

    const data = await res.json();
    const user = tipo === "funcionario" ? data.funcionario : data.usuario;

    if (!user) throw new Error("Usuário não encontrado");

    // Preenche campos básicos
    document.getElementById("fname").value = user.nome || "";
    document.getElementById("email").value = user.email || "";
    document.getElementById("phone").value = user.telefone || "";
document.getElementById("senha").value = user.senha ?? "";

    // Preenche curso e série para alunos
    if (tipo === "usuario") {
      if (user.FK_tipo_usuario_id == 1) { // 1 = aluno
        const cursoEl = document.getElementById("curso");
        const serieEl = document.getElementById("serie");
        if (cursoEl) cursoEl.value = user.nome_curso || "";
        if (serieEl) serieEl.value = user.serie || "";
      }
    }

    // Preenche função para funcionário
    if (tipo === "funcionario") {
      const funcaoEl = document.getElementById("funcao");
      if (funcaoEl) funcaoEl.value = user.FK_funcao_id || "";
    }

  } catch (err) {
    console.error("Erro ao carregar dados:", err);
    alert("Erro ao carregar dados do usuário: " + err.message);
  }
}

// ================ Inicialização ================
const path = window.location.pathname;

if (path.includes("areaAluno.html") || path.includes("areaProf.html")) {
  const usuarioSalvo = JSON.parse(localStorage.getItem("usuario"));
  if (!usuarioSalvo) {
    alert("Usuário não logado!");
    window.location.href = "./index.html";
  } else {
    carregarDados(usuarioSalvo.id, "usuario");
  }
}

if (path.includes("areaAdm.html") || path.includes("areaAdm2.html")) {
  const funcionarioSalvo = JSON.parse(localStorage.getItem("funcionario"));
  if (!funcionarioSalvo) {
    alert("Usuário não logado!");
    window.location.href = "./index.html";
  } else {
    carregarDados(funcionarioSalvo.id, "funcionario");

    const funcaoCampo = document.getElementById("funcao");
    if (funcaoCampo) funcaoCampo.disabled = true; // Apenas visualização
  }
}
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 57 - Código para carregamento de usuário B

```
// ================ Salvar alterações ================
const editarBtn = document.getElementById('btnEditar');

if (editarBtn) {
  editarBtn.addEventListener('click', async (e) => {
    e.preventDefault();

    const usuario = JSON.parse(localStorage.getItem("usuario"));
    const funcionario = JSON.parse(localStorage.getItem("funcionario"));

    let id, tipo;
    if (usuario) { id = usuario.id; tipo = "usuario"; }
    else if (funcionario) { id = funcionario.id; tipo = "funcionario"; }
    else {
      alert("Usuário não logado!");
      window.location.href = "./login.html";
      return;
    }

    const dadosAtualizados = {};
    const telefoneEl = document.getElementById("phone");
    if (telefoneEl && telefoneEl.value.trim() !== "") dadosAtualizados.telefone = telefoneEl.value.trim();

    const senhaEl = document.getElementById("senha");
    if (senhaEl && senhaEl.value.trim() !== "") dadosAtualizados.senha = senhaEl.value.trim();

    const nomeEl = document.getElementById("fname");
    const emailEl = document.getElementById("email");
    if (nomeEl) dadosAtualizados.nome = nomeEl.value.trim();
    if (emailEl) dadosAtualizados.email = emailEl.value.trim();

    if (tipo === "funcionario") {
      const funcaoEl = document.getElementById("funcao");
      if (funcaoEl) dadosAtualizados.FK_funcao_id = funcaoEl.value;
    }

    try {
      const token = localStorage.getItem("token");
      const url = tipo === "funcionario"
        ? `http://localhost:3000/api/funcionarios/${id}`
        : `http://localhost:3000/api/usuarios/${id}`;

      const response = await fetch(url, {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
          "Authorization": "Bearer " + token
        },
        body: JSON.stringify(dadosAtualizados),
      });

      const data = await response.json();
      alert(data.message || "Atualizado com sucesso!");
    } catch (err) {
      console.error("Erro ao atualizar:", err);
      alert("Erro ao atualizar dados.");
    }
  });
}
```

FONTE: AUTORIA PRÓPRIA (2025)

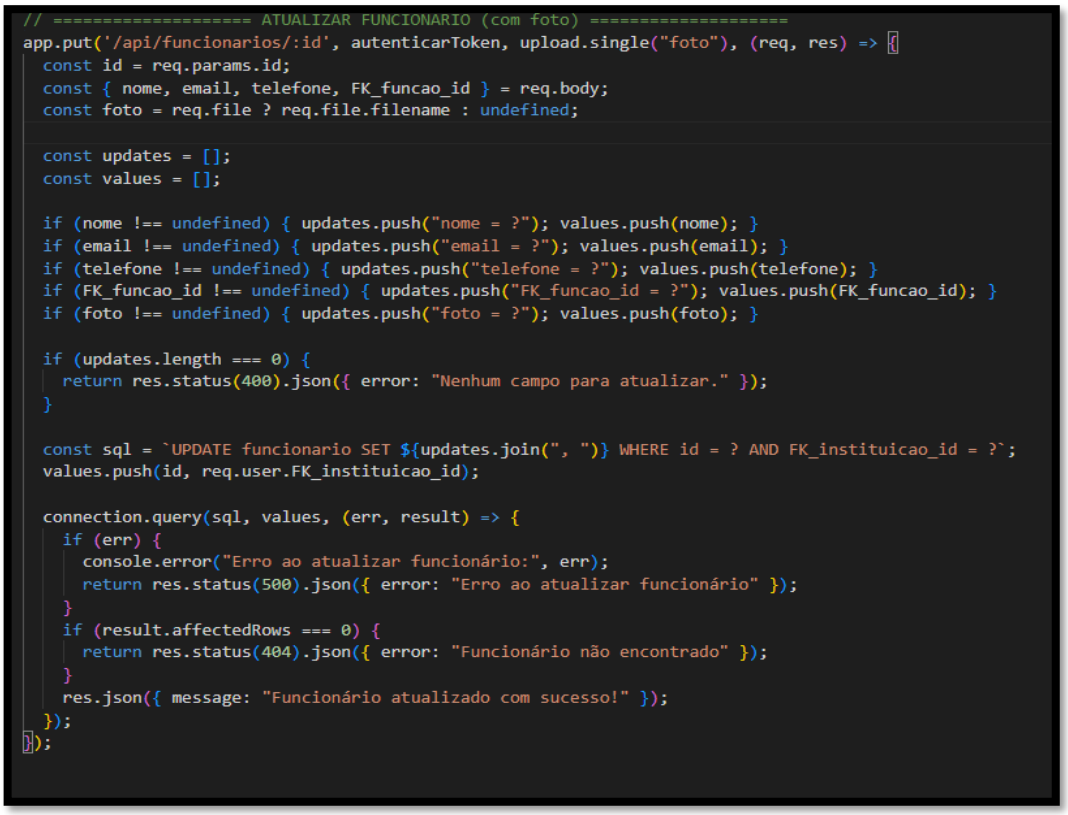
As Figuras 56 e 57 apresentam o código *front-end* responsável por carregar e atualizar informações de usuários, funcionários e administradores no sistema.

Na primeira parte, a função carregarDados utiliza *fetch* para realizar requisições à API e trazer os dados do banco, preenchendo automaticamente os campos da interface como nome, *e-mail*, telefone e senha. Há condições específicas para distinguir se o registro corresponde a um aluno, completando também curso e série, ou a um funcionário, atribuindo sua função. Além disso, o código inclui tratamento de erros para garantir que mensagens adequadas sejam exibidas ao usuário em caso de falha de conexão ou ausência de dados.

Na sequência, é implementado o botão de edição, que coleta os novos valores inseridos nos campos e dispara uma requisição *PUT* para a API. Essa atualização envia apenas os dados realmente modificados, mantendo a integridade do banco. Caso a operação seja concluída com sucesso, o sistema exibe uma mensagem confirmando a alteração. Em situações de erro, o usuário é informado através de alertas.

Assim, esse conjunto garante a integração entre a interface e o servidor, permitindo consultar e editar informações em tempo real, de forma prática e segura.

Figura 58 - Conexão com banco de dados carregamento de usuários



```
// ==================== ATUALIZAR FUNCIONARIO (com foto) ====================
app.put('/api/funcionarios/:id', autenticarToken, upload.single("foto"), (req, res) => {
  const id = req.params.id;
  const { nome, email, telefone, FK_funcao_id } = req.body;
  const foto = req.file ? req.file.filename : undefined;

  const updates = [];
  const values = [];

  if (nome !== undefined) { updates.push("nome = ?"); values.push(nome); }
  if (email !== undefined) { updates.push("email = ?"); values.push(email); }
  if (telefone !== undefined) { updates.push("telefone = ?"); values.push(telefone); }
  if (FK_funcao_id !== undefined) { updates.push("FK_funcao_id = ?"); values.push(FK_funcao_id); }
  if (foto !== undefined) { updates.push("foto = ?"); values.push(foto); }

  if (updates.length === 0) {
    return res.status(400).json({ error: "Nenhum campo para atualizar." });
  }

  const sql = `UPDATE funcionario SET ${updates.join(", ")} WHERE id = ? AND FK_instituicao_id = ?`;
  values.push(id, req.user.FK_instituicao_id);

  connection.query(sql, values, (err, result) => {
    if (err) {
      console.error("Erro ao atualizar funcionário:", err);
      return res.status(500).json({ error: "Erro ao atualizar funcionário" });
    }
    if (result.affectedRows === 0) {
      return res.status(404).json({ error: "Funcionário não encontrado" });
    }
    res.json({ message: "Funcionário atualizado com sucesso!" });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

A Figura 58 mostra o código *back-end* responsável por processar as atualizações enviadas pelo *front-end* para funcionários, administradores e usuários. Após autenticação, os dados recebidos do formulário são validados e apenas os campos realmente alterados são preparados para atualização. Esse processo é feito dinamicamente por meio da construção de uma *query* SQL com os valores fornecidos.

Entre os campos possíveis de atualização estão: nome, *e-mail*, telefone, senha, curso, série, função e até mesmo a foto do perfil, caso enviada. O código também inclui verificações para evitar operações vazias e tratamento de erros como: campos não preenchidos, falhas de conexão com o banco ou inexistência do registro solicitado.

Ao final da execução, o servidor retorna uma resposta clara ao cliente: confirmação de sucesso, erro de processamento ou inexistência do usuário/funcionário. Dessa forma, essas rotas garantem a persistência das edições feitas no sistema, integrando diretamente o *front-end* com o banco de dados de forma confiável e estruturada.

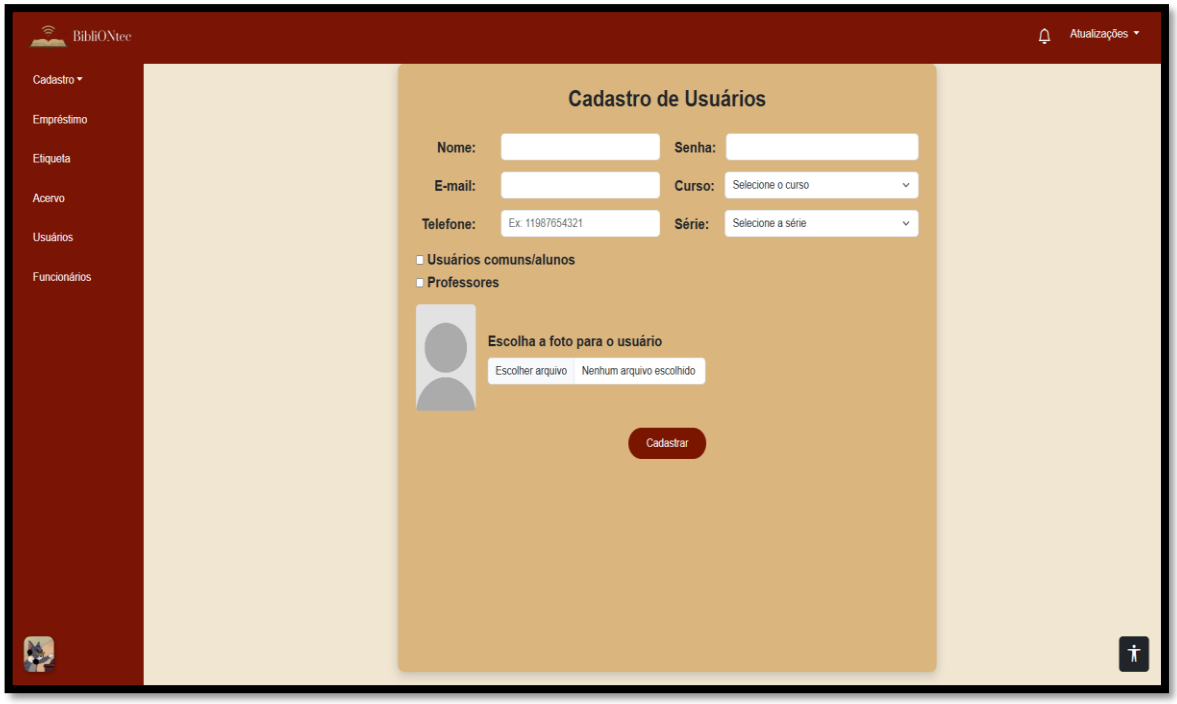
Além disso, a implementação utiliza boas práticas ao adotar uma lógica dinâmica de atualização, permitindo flexibilidade para que apenas os dados modificados sejam alterados no banco, sem sobrescrever informações que permaneceram inalteradas. Isso otimiza a performance e reduz o risco de perda de dados.

Outro aspecto importante é a validação de segurança, já que apenas usuários autenticados podem realizar operações de alteração, evitando acessos indevidos. A manipulação das fotos de perfil também é contemplada, garantindo que arquivos sejam tratados de forma apropriada.

Por fim, o retorno em formato JSON permite que o *front-end* tenha *feedback* imediato e claro, possibilitando exibir mensagens ao usuário e atualizar a interface em tempo real. Esse conjunto de cuidados demonstra a integração sólida entre camadas do sistema e assegura que as informações mantidas no banco de dados estejam sempre consistentes e atualizadas.

3.4.4 – Cadastro de usuários

Figura 59 - Tela de cadastro de usuários



FONTE: AUTORIA PRÓPRIA (2025)

A tela apresentada na figura 59 corresponde ao formulário de cadastro de usuários do sistema BibliONtec. Nela, o administrador ou bibliotecário pode inserir os dados necessários para registrar novos usuários na plataforma.

Os campos disponíveis incluem: nome, *e-mail*, telefone, senha, curso e série, permitindo que o sistema organize as informações de forma estruturada e adequada às necessidades da instituição. Além disso, há opções de seleção para diferenciar o tipo de usuário, podendo ser usuário comum/aluno ou professor, o que facilita a gestão e a atribuição de permissões dentro do sistema.

Outro recurso presente é a possibilidade de inserção de foto de perfil, por meio do *upload* de um arquivo, tornando o cadastro mais completo e personalizado. Ao final, o botão “Cadastrar” conclui o processo e salva os dados no banco de dados.

Essa tela tem como objetivo simplificar o gerenciamento de usuários, oferecendo uma interface intuitiva e organizada, garantindo que todos os registros sejam feitos de forma padronizada e eficiente.

Figura 60 - Código de cadastro de usuários A

```
// ----------- LÓGICA DE CADASTRO DE ALUNO ------------
const formCadastro = document.getElementById('formCadastro');

if (formCadastro) {
  const professorCheckbox = document.getElementById('professor');
  const alunoCheckbox = document.getElementById('aluno');
  const cursoSelect = document.getElementById('curso');
  const serieSelect = document.getElementById('serie');
  const cursoGroup = document.getElementById('cursoGroup');
  const serieGroup = document.getElementById('serieGroup');
  const previewBox = document.querySelector("div[style*='background-color: #5d3b1a']");
  const fotoInput = document.getElementById("foto");
  const funcionarioLogado = JSON.parse(localStorage.getItem("funcionario"));
  const funcionario_id = funcionarioLogado ? funcionarioLogado.id : null;
  function atualizarCampos() {
    const isProfessor = professorCheckbox.checked;

    if (isProfessor) {
      cursoGroup.classList.add('oculto');
      serieGroup.classList.add('oculto');
      cursoSelect.removeAttribute('required');
      serieSelect.removeAttribute('required');
      cursoSelect.value = "";
      serieSelect.value = "";
    } else {
      cursoGroup.classList.remove('oculto');
      serieGroup.classList.remove('oculto');
      cursoSelect.setAttribute('required', 'required');
      serieSelect.setAttribute('required', 'required');
    }
  }


  alunoCheckbox.addEventListener("change", () => {
    if (alunoCheckbox.checked) professorCheckbox.checked = false;
    atualizarCampos();
  });

  professorCheckbox.addEventListener("change", () => {
    if (professorCheckbox.checked) alunoCheckbox.checked = false;
    atualizarCampos();
  });

  fotoInput.addEventListener("change", (e) => {
    const file = e.target.files[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onload = (event) => {
      previewBox.style.backgroundImage = `url('${event.target.result}')`;
      previewBox.style.backgroundSize = "cover";
      previewBox.style.backgroundPosition = "center";
    };
    reader.readAsDataURL(file);
  });
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 61 - Código de cadastro de usuários B



```
atualizarCampos(); // Executa ao carregar a página

formCadastro.addEventListener('submit', async function (event) {
  event.preventDefault();

  const nome = document.getElementById('nome').value;
  const telefone = document.getElementById('telefone').value;
  const email = document.getElementById('email').value;
  const senhaInput = document.getElementById('senha');
  let senha = senhaInput.value;

  if (!senha) {
    senha = gerarSenhaSegura();
    senhaInput.value = senha;
    alert(`Senha gerada automaticamente: ${senha}`);
  } else if (!validarSenha(senha)) {
    alert("Senha inválida! A senha deve conter exatamente 8 caracteres, com pelo menos:\n- 1 letra maiúscula (A-Z)\n- 1 letra minúscula (a-z)\n- 1 número (0-9)");
    return;
  }

  const tipo = alunoCheckbox.checked ? 1 : (professorCheckbox.checked ? 2 : null);
  const foto = fotoInput.files[0];

  if (!nome || !telefone || !email || !senha || !tipo) {
    alert("Preencha todos os campos obrigatórios.");
    return;
  }

  const cursoTexto = cursoSelect.value;
  const serieTexto = serieSelect.value;

  const cursoMap = {
    "Desenvolvimento de Sistemas": 1,
    "Administração": 2,
    "Automação Industrial": 3
  };
  const serieMap = {
    "1ª": 1,
    "2ª": 2,
    "3ª": 3
  };

  const curso_id = cursoMap[cursoTexto];
  const serie = serieMap[serieTexto];

  const formData = new FormData();
  formData.append('nome', nome);
  formData.append('telefone', telefone);
  formData.append('email', email);
  formData.append('senha', senha);
  formData.append('tipo_usuario_id', tipo);
  formData.append('curso_id', curso_id || "");
  formData.append('serie', serie || "");
  formData.append('funcionario_id', funcionario_id || "");
  if (foto) formData.append('foto', foto);



  try {
    const response = await fetch('http://localhost:3000/cadastrarAluno', {
      method: 'POST',
      body: formData
    });

    const contentType = response.headers.get("content-type");

    if (!response.ok) {
      const data = await response.json().catch(() => ({}));
      const erroDiv = document.getElementById('mensagemErroAluno');
      erroDiv.innerText = data.error || 'Erro ao cadastrar aluno.';
      erroDiv.style.display = 'block';
      return;
    } else {
      document.getElementById('mensagemErro').style.display = 'none';
    }


    if (contentType && contentType.includes("application/json")) {
      const data = await response.json();
      alert(data.message || "Usuário cadastrado!");
      formCadastro.reset();
      atualizarCampos();
      previewBox.style.backgroundImage = '';
    } else {
      const text = await response.text();
      console.warn("Resposta inesperada:", text);
      alert("Resposta inesperada do servidor.");
    }
  } catch (error) {
    console.error("Erro ao enviar dados:", error);
    alert("Erro de rede ou servidor!");
  }

});
}

document.querySelectorAll('.apenasTexto').forEach(input => {
  input.addEventListener('input', () => {
    input.value = input.value.replace(/[^a-zA-ZÀ-ÿ\s]/g, '');
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

As figuras 60 e 61 apresentam o código responsável pela lógica de cadastro de alunos e professores no sistema. Ele começa identificando os elementos do formulário de cadastro, como nome, telefone, *e-mail*, senha, curso, série, foto e as opções para selecionar se o usuário é aluno ou professor. A partir disso, o sistema ajusta automaticamente os campos que devem aparecer ou ser obrigatórios. Por exemplo, quando o usuário escolhe a opção professor, os campos de curso e série são ocultados e deixam de ser obrigatórios, já quando a opção aluno é selecionada, esses campos são exibidos e passam a ser exigidos para concluir o cadastro. Além disso, o código impede que as duas opções sejam selecionadas ao mesmo tempo, garantindo consistência no preenchimento.

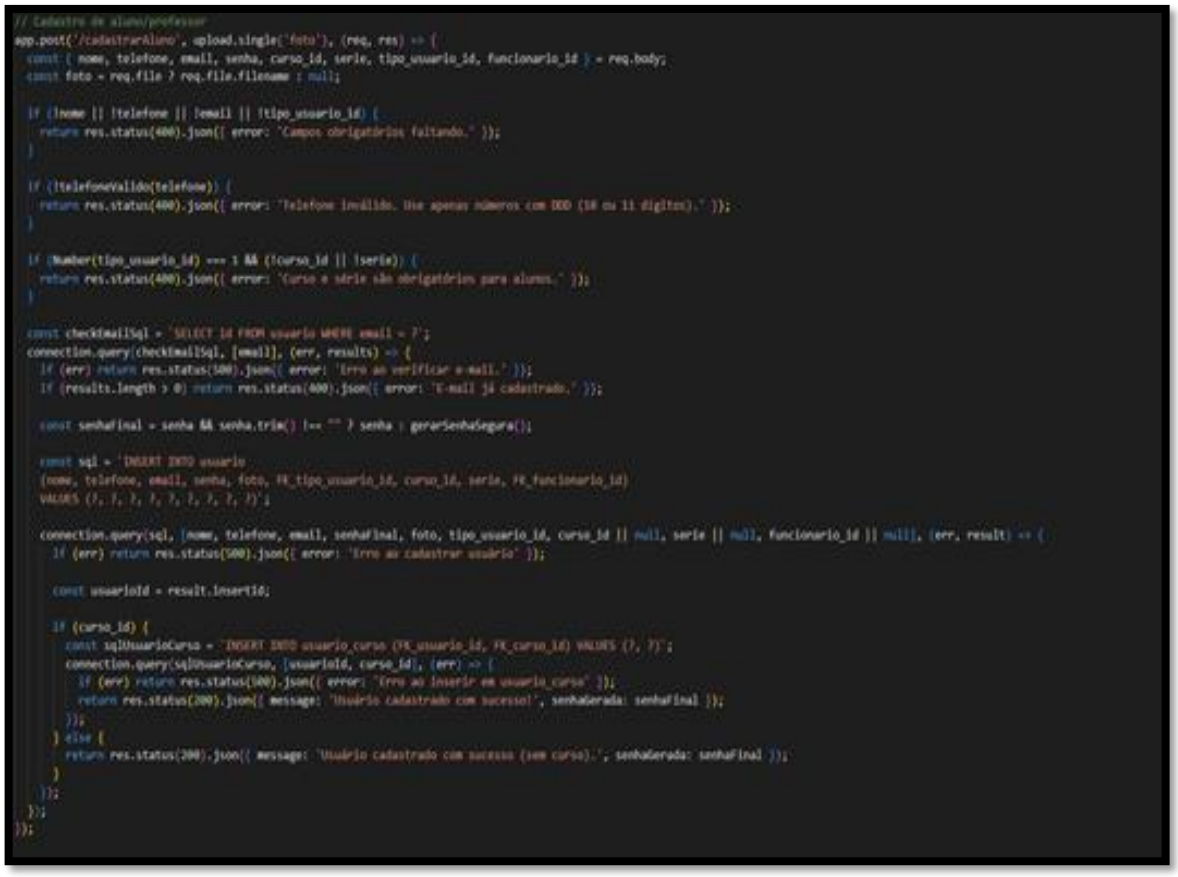
Outro ponto importante é o *upload* da foto de perfil. O sistema permite que o usuário selecione uma imagem e, assim que isso acontece, a foto já aparece como pré-visualização dentro do formulário. Também há uma etapa de validação da senha, que deve ter pelo menos oito caracteres, incluindo uma letra maiúscula, uma minúscula e um número. Caso esses requisitos não sejam atendidos, o sistema interrompe o processo e exibe uma mensagem de alerta.

Depois das validações, as informações fornecidas pelo usuário são organizadas em um objeto chamado *FormData*, que reúne os dados pessoais (nome, telefone, *e-mail* e senha), o tipo de usuário (aluno ou professor), curso e série quando necessário, foto de perfil e também a identificação do funcionário responsável pelo cadastro. Esses dados são então enviados para o servidor por meio de uma requisição do tipo *POST* para o endereço http://localhost:3000/cadastrarAluno.

Quando o servidor recebe essas informações, ele as processa e retorna uma resposta. Caso o cadastro seja realizado com sucesso, o sistema exibe uma mensagem de confirmação para o usuário. Caso ocorra algum erro, o código apresenta uma mensagem de falha. Além disso, o campo de telefone passa por um tratamento especial para aceitar apenas caracteres válidos, evitando que informações incorretas sejam enviadas.

Dessa forma, esse código funciona como uma ponte entre o formulário que o usuário preenche e o banco de dados do sistema, pois ele coleta, valida e organiza as informações antes de enviá-las para serem armazenadas.

Figura 62 - Conexão com banco de dados cadastro de usuário



```
// Cadastro de aluno/professor
app.post('/cadastrarAluno', upload.single('foto'), (req, res) => {
  const { nome, telefone, email, senha, curso_id, serie, tipo_usuario_id, funcionario_id } = req.body;
  const foto = req.file ? req.file.filename : null;

  if (!nome || !telefone || !email || !tipo_usuario_id) {
    return res.status(400).json({ error: 'Campos obrigatórios faltando.' });
  }

  if (!telefoneValido(telefone)) {
    return res.status(400).json({ error: 'Telefone inválido. Use apenas números com DDD (10 ou 11 dígitos).' });
  }

  if (Number(tipo_usuario_id) === 1 && (!curso_id || !serie)) {
    return res.status(400).json({ error: 'Curso e série são obrigatórios para alunos.' });
  }

  const checkEmailSql = 'SELECT id FROM usuario WHERE email = ?';
  connection.query(checkEmailSql, [email], (err, results) => {
    if (err) return res.status(500).json({ error: 'Erro ao verificar e-mail.' });
    if (results.length > 0) return res.status(400).json({ error: 'E-mail já cadastrado.' });

    const senhaFinal = senha && senha.trim() !== "" ? senha : gerarSenhaSegura();

    const sql = `INSERT INTO usuario
    (nome, telefone, email, senha, foto, FK_tipo_usuario_id, curso_id, serie, FK_funcionario_id)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)`;

    connection.query(sql, [nome, telefone, email, senhaFinal, foto, tipo_usuario_id, curso_id || null, serie || null, funcionario_id || null], (err, result) => {
      if (err) return res.status(500).json({ error: 'Erro ao cadastrar usuário' });

      const usuarioId = result.insertId;

      if (curso_id) {
        const sqlUsuarioCurso = `INSERT INTO usuario_curso (FK_usuario_id, FK_curso_id) VALUES (?, ?)`;
        connection.query(sqlUsuarioCurso, [usuarioId, curso_id], (err) => {
          if (err) return res.status(500).json({ error: 'Erro ao inserir em usuario_curso' });
          return res.status(200).json({ message: 'Usuário cadastrado com sucesso!', senhaGerada: senhaFinal });
        });
      } else {
        return res.status(200).json({ message: 'Usuário cadastrado com sucesso (sem curso).', senhaGerada: senhaFinal });
      }
    });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

O trecho de código apresentado na figura 62 corresponde à rota de cadastro de alunos e professores no sistema BibliONtec, utilizando a linguagem JavaScript com Node.js e integração com banco de dados MySQL.

A função é definida como uma requisição do tipo *POST* (*app*.*post*(“/cadastrarAluno”, ...)), responsável por receber os dados enviados pelo formulário de cadastro, como nome, telefone, *e-mail*, senha, curso, série, tipo de usuário e funcionário responsável. Além disso, também há suporte para o *upload* de uma foto de perfil, que é salva no servidor.

O código realiza diversas validações antes de inserir os dados no banco, garantindo a integridade das informações. Entre essas validações estão:

Verificação de campos obrigatórios (nome, telefone, *e-mail* e tipo de usuário).

Validação do telefone, permitindo apenas números com DDD.

Obrigatoriedade de curso e série quando o tipo de usuário for aluno.

Checagem no banco de dados para confirmar se o *e-mail* já está cadastrado.

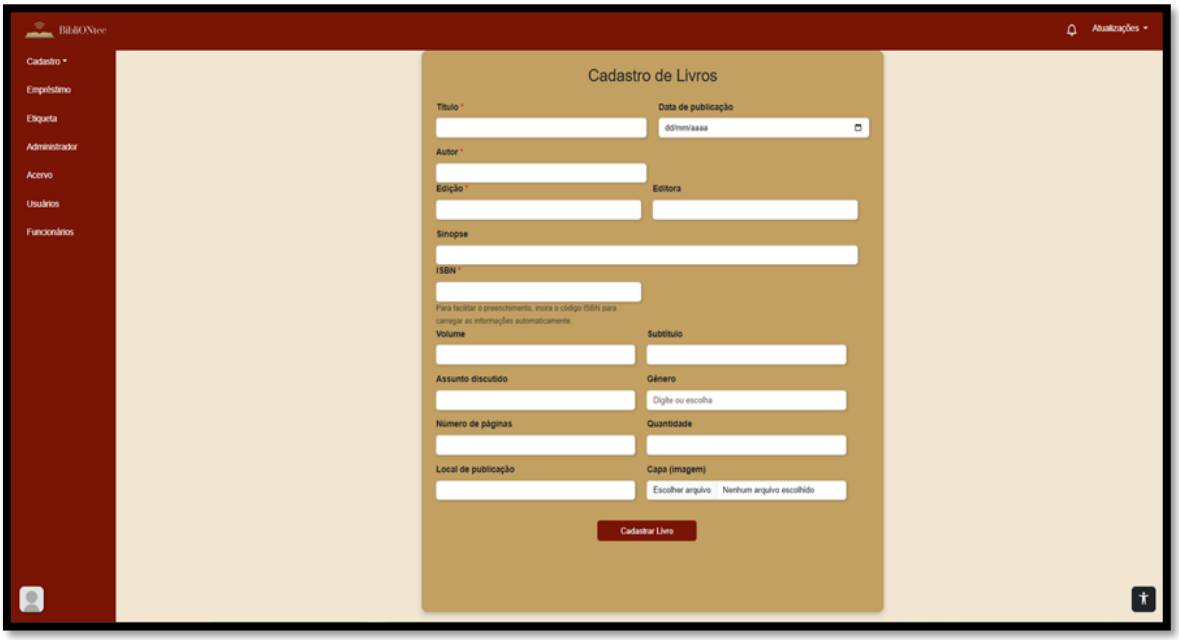
Se o cadastro for válido, o sistema gera ou utiliza a senha informada, insere o novo usuário na tabela usuario e, caso haja vínculo com curso, também registra a relação na tabela usuario_curso.

Ao final, a resposta da API retorna uma mensagem em JSON, indicando se o cadastro foi realizado com sucesso ou se ocorreu algum erro, informando inclusive a senha gerada automaticamente quando necessário.

Esse código é essencial para garantir que o processo de cadastro seja seguro, confiável e bem estruturado, evitando duplicidade de usuários e assegurando a consistência dos dados no sistema.

3.4.5 – Cadastro de livros

Figura 63 - Tela cadastro de livros



FONTE: AUTORIA PRÓPRIA (2025)

A tela apresentada na figura 63 corresponde ao Cadastro de Livros no sistema BibliONtec. Essa interface foi desenvolvida com o objetivo de permitir que o funcionário e bibliotecário registre as obras disponíveis no acervo de maneira simples e intuitiva, garantindo um controle organizado, preciso e detalhado das informações relacionadas a cada livro. A tela é projetada para atender às necessidades tanto de

registros rápidos quanto de consultas mais aprofundadas, facilitando o processo de gerenciamento do acervo de forma eficaz.

Os campos disponibilizados abrangem os principais dados necessários para a identificação de uma obra, como título, autor, edição, editora, data de publicação e ISBN, que são obrigatórios para validar o cadastro. As informações são preenchidas automaticamente caso o ISBN informado esteja disponível. Esses dados fundamentais são essenciais para garantir que a obra seja identificada de maneira única e precisa, evitando confusões no futuro. Além desses campos obrigatórios, o sistema oferece uma série de campos complementares como sinopse, volume, subtítulo, assunto discutido, gênero, número de páginas, local de publicação e quantidade de exemplares. Esses campos permitem que o sistema registre tanto dados técnicos, como a edição e o número de páginas, quanto dados descritivos, como a sinopse e o gênero literário, fornecendo uma visão mais completa e rica sobre cada livro.

A inclusão desses dados complementares favorece a consulta posterior pelos usuários, otimizando tanto a busca pelo conteúdo quanto o gerenciamento do acervo. Os usuários podem encontrar rapidamente livros que atendem às suas necessidades específicas. Essa variedade de informações permite uma busca refinada, o que torna o sistema mais eficiente e acessível para os bibliotecários e leitores.

Outro recurso disponível é o *upload* da capa do livro, que pode ser facilmente inserido no sistema por meio da seleção de uma imagem armazenada no dispositivo.

Essa funcionalidade não apenas torna o catálogo mais visual e agradável, mas também facilita a identificação das obras pelos leitores, principalmente em um acervo grande e diversificado.

Após o preenchimento de todos os campos obrigatórios e complementares, o Bibliotecário pode clicar no botão Cadastrar Livro, que conclui o processo e envia os dados para o banco de dados. Esse procedimento torna a obra oficialmente registrada no sistema, o que permite que ela seja acessada para empréstimos, consultas e outras operações futuras. A validação e a verificação dos dados são realizadas automaticamente pelo sistema, garantindo que todos os campos obrigatórios sejam preenchidos corretamente antes do envio.

Essa tela, portanto, desempenha um papel essencial na manutenção do acervo da biblioteca. O registro adequado de dados reflete diretamente na eficiência das consultas, na organização do acervo e no controle do patrimônio da biblioteca, melhorando a experiência do usuário e a produtividade dos administradores. Além disso, a interface intuitiva e funcional ajuda a otimizar o tempo dos bibliotecários, permitindo que o processo de cadastro seja realizado de forma rápida e precisa.

Figura 64 - Código cadastro de livros A



```
document.addEventListener('DOMContentLoaded', async () => {
  const loadingIsbn = document.getElementById('loadingIsbn');
  const token = localStorage.getItem('token');
  const usuarioId = localStorage.getItem('usuarioId'); // Id do funcionário logado

  if (!token) {
    alert('Você precisa estar logado para cadastrar livros.');
    return;
  }

  // Inputs principais
  const form = document.getElementById('formLivro');
  const inputGeneroTexto = document.getElementById('generoTexto');
  const inputGeneroId = document.getElementById('FK_genero_id');
  const isbnInput = document.getElementById('isbnInput');
  const autorInput = document.getElementById('autorTexto');
  const editoraInput = document.getElementById('editoraTexto');

  let autoresExistentes = [];
  let editorasExistentes = [];
  let generosExistentes = await carregarGeneros();

  function mostrarLoading() { if (loadingIsbn) loadingIsbn.style.display = 'inline-block'; }
  function esconderLoading() { if (loadingIsbn) loadingIsbn.style.display = 'none'; }

  // Associar input de gênero ao ID
  inputGeneroTexto.addEventListener('input', () => {
    const texto = inputGeneroTexto.value.trim().toLowerCase();
    const encontrado = generosExistentes.find(g => g.genero.toLowerCase() === texto);
    if (inputGeneroId) inputGeneroId.value = encontrado ? encontrado.id : '';
  });

  // Autocomplete autor
  autorInput.addEventListener('input', () => {
    const texto = autorInput.value.trim().toLowerCase();
    const encontrado = autoresExistentes.find(a => a.nome.toLowerCase() === texto);
    const inputAutorId = document.getElementById('FK_autor_id');
    if (inputAutorId) inputAutorId.value = encontrado ? encontrado.id : '';
  });

  // Autocomplete editora
  editoraInput.addEventListener('input', () => {
    const texto = editoraInput.value.trim().toLowerCase();
    const encontrado = editorasExistentes.find(e => e.editora.toLowerCase() === texto);
    const inputEditoraId = document.getElementById('FK_editora_id');
    if (inputEditoraId) inputEditoraId.value = encontrado ? encontrado.id : '';
  });

  // Preenchimento via ISBN
  isbnInput.addEventListener('blur', async () => {
    const isbn = isbnInput.value.trim();
    if (!isbn) return;
    mostrarLoading();

    try {
      const response = await fetch(`https://www.googleapis.com/books/v1/volumes?q=isbn:${isbn}&langRestrict=pt&country=BR`);
      const data = await response.json();

      if (data.totalItems > 0) {
        const livro = data.items[0].volumeInfo;

        const traduzir = async (texto) => {
          if (!texto) return '';
          try {
            const url = `https://translate.googleapis.com/translate_a/single?client=gtx&sl=en&tl=pt&dt=t&q=${encodeURIComponent(texto)}`;
            const resp = await fetch(url);
            const traduzido = await resp.json();
            return traduzido[0][0][0];
          } catch { return texto; }
        };

        // Preenche campos
        const tituloInput = document.querySelector('input[name="titulo"]');
        if (tituloInput) tituloInput.value = await traduzir(livro.title);

        const subtituloInput = document.querySelector('input[name="subtitulo"]');
        if (subtituloInput) subtituloInput.value = await traduzir(livro.subtitle);

        const descricaoPT = await traduzir(livro.description);
        const editoraPT = await traduzir(livro.publisher);
        const generoPT = await traduzir(livro.categories?.join(', ') || '');

        if (autorInput) autorInput.value = livro.authors?.join(', ') || '';
        if (editoraInput) editoraInput.value = editoraPT;
        if (inputGeneroTexto) inputGeneroTexto.value = generoPT;
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 65 - Código de cadastro de livros B



```
      const dataInput = document.querySelector('input[name="data_publicacao"]');
      if (dataInput) dataInput.value = ajustarData(livro.publishedDate);

      const sinopseInput = document.querySelector('input[name="sinopse"]');
      if (sinopseInput) sinopseInput.value = descricaoPT;

      const paginasInput = document.querySelector('input[name="paginas"]');
      if (paginasInput) paginasInput.value = livro.pageCount || '';

      const volumeInput = document.querySelector('input[name="volume"]');
      if (volumeInput) volumeInput.value = '1';

      const edicaoInput = document.querySelector('input[name="edicao"]');
      if (edicaoInput) edicaoInput.value = '1ª';

      // Buscar ID do gênero se existir
      const generoEncontrado = generosExistentes.find(g => g.genero.toLowerCase() === generoPT.toLowerCase());
      if (inputGeneroId) inputGeneroId.value = generoEncontrado ? generoEncontrado.id : '';

    } else {
      alert('Livro não encontrado na API.');
    }
  } catch (error) {
    console.error('Erro ao consultar API do Google Books:', error);
    alert('Erro ao consultar API.');
  } finally {
    esconderLoading();
  }
});

// Submissão do formulário
form.addEventListener('submit', async (evento) => {
  evento.preventDefault();

  const textoGenero = inputGeneroTexto.value.trim();
  let idGenero = inputGeneroId.value;

  // Se gênero não existe, cadastra
  if (!idGenero && textoGenero !== '') {
    try {
      const resp = await fetch('http://localhost:3000/generos', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json', 'Authorization': 'Bearer ' + token },
        body: JSON.stringify({ genero: textoGenero })
      });

      if (!resp.ok) throw new Error('Erro ao salvar novo gênero');

      generosExistentes = await carregarGeneros();
      const novo = generosExistentes.find(g => g.genero.toLowerCase() === textoGenero.toLowerCase());
      inputGeneroId.value = novo?.id || '';
    } catch (err) {
      alert('Erro ao salvar gênero: ' + err.message);
      return;
    }
  }

  // Se autor não existe, cadastra
  let autorId = document.getElementById('FK_autor_id').value;
  if (!autorId && autorInput.value.trim() !== '') {
    try {
      const resp = await fetch('http://localhost:3000/autores', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json', 'Authorization': 'Bearer ' + token },
        body: JSON.stringify({ nome: autorInput.value.trim() })
      });
      if (!resp.ok) throw new Error('Erro ao salvar autor');
      autoresExistentes = await carregarAutores();
      const novo = autoresExistentes.find(a => a.nome.toLowerCase() === autorInput.value.trim().toLowerCase());
      autorId = novo?.id;
    } catch (err) {
      alert('Erro ao salvar autor: ' + err.message);
      return;
    }
  }

  // Se editora não existe, cadastra
  let editoraId = document.getElementById('FK_editora_id').value;
  if (!editoraId && editoraInput.value.trim() !== '') {
    try {
      const resp = await fetch('http://localhost:3000/editoras', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json', 'Authorization': 'Bearer ' + token },
        body: JSON.stringify({ nome: editoraInput.value.trim() })
      });
```

FONTE: AUTORIA PRÓPRIA (2025)

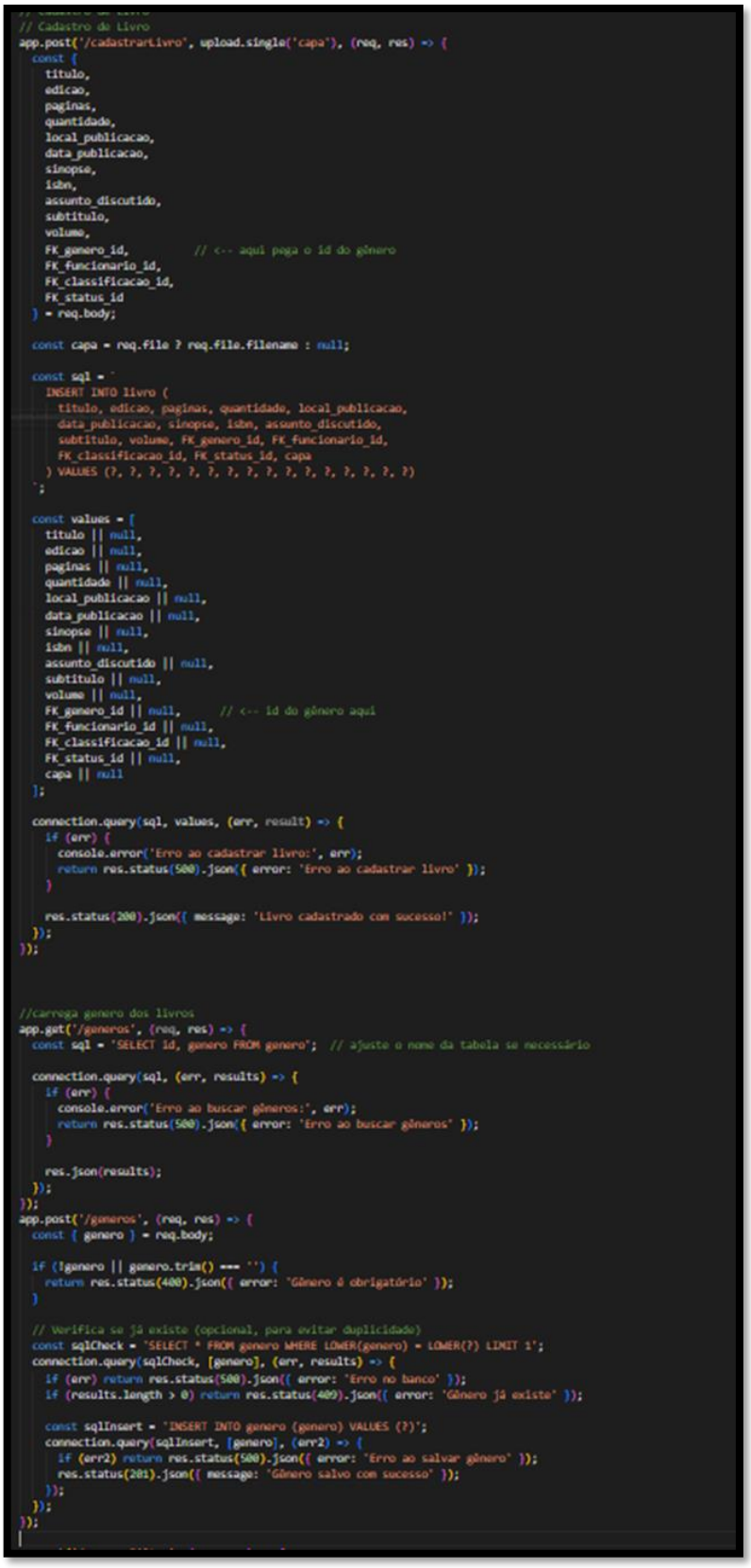
O código apresentado nas figuras 64 e 65 é responsável pela lógica de cadastro de livros em um sistema *web*, utilizando integração com APIs externas e dados do banco local para automatizar o preenchimento e a validação dos campos do formulário. Logo ao carregar a página, o sistema identifica os elementos HTML, incluindo os inputs de ISBN, autor, editora e gênero, além do formulário principal de cadastro. Para melhorar a experiência do usuário, há um indicador de carregamento exibido e ocultado durante as requisições assíncronas, fornecendo *feedback* imediato sobre a execução das operações em segundo plano.

Ao digitar informações nos campos de autor, editora ou gênero, o código apresenta listas de seleção preenchidas com os registros já existentes no banco de dados, permitindo que o usuário escolha diretamente o item desejado. Essa associação automática garante que o identificador correto (ID) seja aplicado ao formulário e evita duplicidades na base de dados. Caso o valor digitado não seja encontrado na lista exibida, o sistema realiza uma requisição *POST* ao *back-end*, cadastrando o novo gênero, autor ou editora, e atualiza a lista carregada, de modo que o item fique disponível para futuras utilizações sem necessidade de recarregar a página.

A funcionalidade central está ligada ao evento de "*blur*" do campo ISBN, que ao perder o foco consulta a API do Google Books e retorna as informações do livro encontrado. Caso haja resultados, dados como título, subtítulo, autores, descrição, editora, categorias, data de publicação e número de páginas são automaticamente preenchidos no formulário. Para assegurar que o usuário visualize os dados no idioma adequado, o código integra chamadas à API do Google Translate, traduzindo dinamicamente informações textuais. Caso a tradução falhe, o sistema mantém o valor original, preservando a integridade da informação recebida e evitando falhas no processo de preenchimento automático.

Além disso, a formatação da data é ajustada para o padrão internacional ISO (yyyy-mm-dd), garantindo compatibilidade com o banco e integridade das operações REST (Representational State Transfer – Transferência de Estado Representacional). Ao enviar o formulário, os dados organizados são transmitidos via requisição *POST* ao *endpoint* responsável por persistir o livro no banco de dados.

Figura 66 - Conexão com banco de dados cadastro de livros A



```
// Cadastro de Livro
app.post('/cadastrarLivro', upload.single('capa'), (req, res) => {
  const {
    titulo,
    edicao,
    paginas,
    quantidade,
    local_publicacao,
    data_publicacao,
    sinopse,
    isbn,
    assunto_discutido,
    subtitulo,
    volume,
    FK_genero_id,          // <-- aqui pega o id do gênero
    FK_funcionario_id,
    FK_classificacao_id,
    FK_status_id
  } = req.body;

  const capa = req.file ? req.file.filename : null;

  const sql = `
    INSERT INTO livro (
      titulo, edicao, paginas, quantidade, local_publicacao,
      data_publicacao, sinopse, isbn, assunto_discutido,
      subtitulo, volume, FK_genero_id, FK_funcionario_id,
      FK_classificacao_id, FK_status_id, capa
    ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
  `;

  const values = [
    titulo || null,
    edicao || null,
    paginas || null,
    quantidade || null,
    local_publicacao || null,
    data_publicacao || null,
    sinopse || null,
    isbn || null,
    assunto_discutido || null,
    subtitulo || null,
    volume || null,
    FK_genero_id || null,      // <-- id do gênero aqui
    FK_funcionario_id || null,
    FK_classificacao_id || null,
    FK_status_id || null,
    capa || null
  ];

  connection.query(sql, values, (err, result) => {
    if (err) {
      console.error('Erro ao cadastrar livro:', err);
      return res.status(500).json({ error: 'Erro ao cadastrar livro' });
    }

    res.status(200).json({ message: 'Livro cadastrado com sucesso!' });
  });
});



//carrega genero dos livros
app.get('/generos', (req, res) => {
  const sql = 'SELECT id, genero FROM genero';  // ajuste o nome da tabela se necessário

  connection.query(sql, (err, results) => {
    if (err) {
      console.error('Erro ao buscar gêneros:', err);
      return res.status(500).json({ error: 'Erro ao buscar gêneros' });
    }

    res.json(results);
  });
});
app.post('/generos', (req, res) => {
  const { genero } = req.body;

  if (!genero || genero.trim() === '') {
    return res.status(400).json({ error: 'Gênero é obrigatório' });
  }

  // Verifica se já existe (opcional, para evitar duplicidade)
  const sqlCheck = 'SELECT * FROM genero WHERE LOWER(genero) = LOWER(?) LIMIT 1';
  connection.query(sqlCheck, [genero], (err, results) => {
    if (err) return res.status(500).json({ error: 'Erro no banco' });
    if (results.length > 0) return res.status(409).json({ error: 'Gênero já existe' });

    const sqlInsert = 'INSERT INTO genero (genero) VALUES (?)';
    connection.query(sqlInsert, [genero], (err2) => {
      if (err2) return res.status(500).json({ error: 'Erro ao salvar gênero' });
      res.status(201).json({ message: 'Gênero salvo com sucesso' });
    });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 67 - Conexão com banco de dados cadastro de livros B



```
// ===== Listar autores =====
app.get('/autores', autenticarToken, (req, res) => {
  const sql = 'SELECT id, nome FROM autor ORDER BY nome ASC';
  connection.query(sql, (err, results) => {
    if (err) return res.status(500).json({ error: 'Erro ao buscar autores' });
    res.json(results);
  });
});

// ===== Cadastrar autor =====
app.post('/autores', autenticarToken, (req, res) => {
  const { nome } = req.body;

  if (!nome || nome.trim() === '') {
    return res.status(400).json({ error: 'Nome do autor é obrigatório' });
  }

  // Verifica se já existe
  const sqlCheck = 'SELECT * FROM autor WHERE LOWER(nome) = LOWER(?) LIMIT 1';
  connection.query(sqlCheck, [nome.trim()], (err, results) => {
    if (err) return res.status(500).json({ error: 'Erro no banco ao verificar autor' });
    if (results.length > 0) return res.status(409).json({ error: 'Autor já existe' });

    // Insere novo autor
    const sqlInsert = 'INSERT INTO autor (nome) VALUES (?)';
    connection.query(sqlInsert, [nome.trim()], (err2) => {
      if (err2) return res.status(500).json({ error: 'Erro ao salvar autor' });
      res.status(201).json({ message: 'Autor cadastrado com sucesso' });
    });
  });
});

// ===== Listar editoras =====
app.get('/editoras', autenticarToken, (req, res) => {
  const sql = 'SELECT id, editora AS editora FROM editora ORDER BY editora ASC'; // <-- CORRIGIDO
  connection.query(sql, (err, results) => {
    if (err) {
      console.error('Erro ao buscar editoras:', err);
      return res.status(500).json({ error: 'Erro ao buscar editoras' });
    }
    res.json(results);
  });
});

// ===== Cadastrar editora =====
app.post('/editoras', autenticarToken, (req, res) => {
  const { nome } = req.body;

  if (!nome || nome.trim() === '') {
    return res.status(400).json({ error: 'Nome da editora é obrigatório' });
  }

  const sqlCheck = 'SELECT * FROM editora WHERE LOWER(editora) = LOWER(?) LIMIT 1'; // <-- CORRIGIDO
  connection.query(sqlCheck, [nome.trim()], (err, results) => {
    if (err) return res.status(500).json({ error: 'Erro no banco ao verificar editora' });
    if (results.length > 0) return res.status(409).json({ error: 'Editora já existe' });

    const sqlInsert = 'INSERT INTO editora (editora) VALUES (?)'; // <-- CORRIGIDO
    connection.query(sqlInsert, [nome.trim()], (err2) => {
      if (err2) return res.status(500).json({ error: 'Erro ao salvar editora' });
      res.status(201).json({ message: 'Editora cadastrada com sucesso' });
    });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

O código apresentado nas Figuras 66 e 67 é desenvolvido em Node.js utilizando o *framework Express* e tem como objetivo realizar a comunicação entre o sistema de cadastro de livros e o banco de dados. Ele implementa rotas responsáveis por inserir e consultar informações relacionadas a livros e gêneros.

A primeira rota, *app.post*('/cadastrarLivro', ...), é destinada ao cadastro de livros. Ela utiliza o *middleware upload.single*('capa'), o qual permite o envio de um arquivo associado à capa do livro. Os dados recebidos do formulário são armazenados em constantes, incluindo informações como título, edição, número de páginas, quantidade, local de publicação, data, sinopse, ISBN, subtítulo, volume, além de chaves estrangeiras que relacionam o livro ao gênero, funcionário responsável, classificação e *status*. O arquivo da capa, quando enviado, é tratado e seu nome é salvo na variável capa.

Para inserir os dados no banco, é construída uma instrução SQL do tipo *INSERT*, que insere os valores recebidos na tabela livro. A execução da *query* é feita com *connection.query*(), e, caso ocorra algum erro, uma mensagem de falha é exibida e o servidor retorna o *status* HTTP 500. Se a operação for concluída com sucesso, o sistema envia uma resposta de sucesso com *status* 200 e uma mensagem informando que o livro foi cadastrado corretamente.

Na sequência, o código define a rota *app.get*('/generos', ...), que tem a função de consultar e retornar todos os gêneros cadastrados. Essa rota executa uma *query* SQL do tipo *SELECT*, recuperando o identificador e o nome de cada gênero. Os resultados são enviados em formato JSON para o cliente. Em caso de falha na execução da consulta, o servidor retorna uma mensagem de erro com *status* 500.

Por fim, o código implementa a rota *app.post*('/generos', ...), responsável por cadastrar novos gêneros no banco de dados. Inicialmente, é feita uma validação para verificar se o campo gênero foi enviado corretamente. Caso contrário, o servidor responde com *status* 400 e uma mensagem de erro informando que o gênero é obrigatório. Para evitar duplicidade, antes da inserção é realizada uma consulta SQL *SELECT* que verifica se o gênero já existe no banco, desconsiderando diferenças de maiúsculas e minúsculas. Se o gênero já estiver cadastrado, o servidor retorna uma mensagem informando a duplicidade. Caso contrário, é executada a instrução

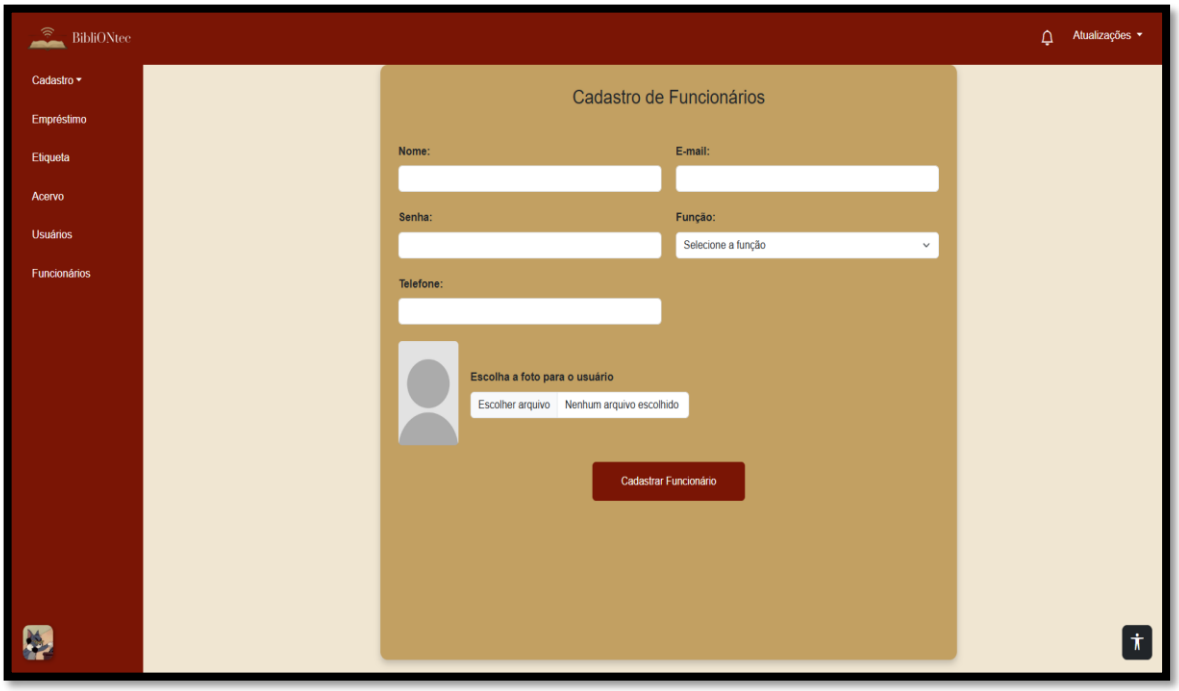
INSERT INTO gênero, registrando o novo gênero. Se o processo ocorrer sem falhas, a resposta enviada ao cliente é uma confirmação de sucesso, com *status* 201.

Além dessas rotas, o sistema também implementa operações voltadas para autores e editoras, igualmente necessárias no contexto do cadastro de livros. A rota *app.get*('/autores', ...) recupera todos os autores cadastrados no banco de dados, listando-os em ordem alfabética. Já a rota *app.post*('/autores', ...) permite o cadastro de um novo autor, verificando previamente se o nome foi informado e se o autor já não existe no banco. Caso as validações sejam atendidas, a inserção é realizada e o servidor responde com *status* 201 e uma mensagem de sucesso.

De forma semelhante, existem as rotas *app.get*('/editoras', ...) e *app.post*('/editoras', ...), responsáveis por listar e cadastrar editoras. O funcionamento é análogo ao dos autores: primeiro são retornados os registros existentes, e no caso do cadastro é feita uma checagem para garantir que o campo obrigatório foi preenchido e que não exista duplicidade. Em caso de sucesso, a nova editora é inserida e o sistema confirma a operação com *status* 201, exibindo uma mensagem de sucesso.

3.4.6 – Cadastro de funcionários

Figura 68 - Tela de cadastro de funcionários



FONTE: AUTORIA PRÓPRIA (2025)

A tela de cadastro de funcionários apresentada na figura 68 mostra um formulário que possibilita a inserção dos dados principais de um colaborador do sistema, incluindo nome, *e-mail*, senha, telefone, função e foto. O *layout* mantém a identidade visual do sistema, com campos organizados de forma clara e objetiva, garantindo acessibilidade e usabilidade. Além disso, há um botão centralizado para efetivar o cadastro, reforçando a simplicidade da navegação.

Figura 69 - Código cadastro de funcionários

```
// ---------- LÓGICA DE CADASTRO DE FUNCIONÁRIO ----------
const formFunc = document.getElementById('formFuncionario');
if (formFunc) {
  formFunc.addEventListener('submit', async function (event) {
    event.preventDefault();

    console.log("Formulário de funcionário enviado!"); // DEBUG

    const nome = document.getElementById('nome').value;
    const senhaInput = document.getElementById('senha');
    let senha = senhaInput.value;

    if (!senha) {
      senha = gerarSenhaSegura();
      senhaInput.value = senha;
      alert(`Senha gerada automaticamente: ${senha}`);
    } else if (!validarSenha(senha)) {
      alert("Senha inválida! A senha deve conter pelo menos 8 caracteres, com pelo menos:\n- 1 letra maiúscula (A-Z)\n- 1 letra minúscula (a-z)\n- 1 número (0-9)");
      return;
    }

    const email = document.getElementById('email').value;
    const foto = document.getElementById('foto').files[0];
    const telefone = document.getElementById('telefone').value;
    const funcionarioLogado = JSON.parse(localStorage.getItem("funcionario"));
    const funcionario_id = funcionarioLogado ? funcionarioLogado.id : null;
    const permissoesSelecionadas = [...document.querySelectorAll('input[name="permissao"]:checked')]
      .map(el => el.value);
    const funcao_id_el = document.getElementById('funcao');
    const funcao_id = funcao_id_el ? funcao_id_el.value : null;

    if (!nome || !senha || !email) {
      alert("Preencha todos os campos obrigatórios.");
      return;
    }

    const formData = new FormData();
    formData.append('nome', nome);
    formData.append('senha', senha);
    formData.append('email', email);
    formData.append('telefone', telefone);
    if (foto) formData.append('foto', foto);
    if (funcao_id) formData.append('FK_funcao_id', funcao_id);


    permissoesSelecionadas.forEach(p => formData.append('permissoes[]', p));

    try {
      const response = await fetch('http://localhost:3000/cadastrarFuncionario', {
        method: 'POST',
        body: formData
      });

      const data = await response.json().catch(() => ({}));

      if (!response.ok) {
        const erroDiv = getOrCreateErrorBox('mensagemErroFuncionario', formFunc);

        erroDiv.innerText = data.error || 'Erro ao cadastrar funcionário.';
        erroDiv.style.display = 'block';
        return;
      }

      alert(data.message || 'Funcionário cadastrado com sucesso!');
      formFunc.reset();
    } catch (error) {
      erroDiv.innerText = 'Erro ao enviar o formulário. Tente novamente.';
      erroDiv.style.display = 'block';
    }
  });
}
```

FONTE: AUTORIA PRÓPRIA (2025)

A figura 69 apresenta o código JavaScript responsável pela lógica do cadastro de funcionários captura os dados inseridos pelo usuário no formulário e executa validações antes do envio. A senha pode ser digitada ou gerada automaticamente de forma segura, garantindo requisitos mínimos de complexidade. Caso haja campos obrigatórios não preenchidos ou senha inválida, o sistema emite alertas para o

usuário. Após a validação, os dados são encapsulados em um objeto *FormData*, permitindo inclusive o envio de arquivos (como a foto do funcionário). Em seguida, a aplicação utiliza a função *fetch*() para realizar uma requisição HTTP do tipo *POST* para a rota da API (/cadastrarFuncionario), enviando as informações ao servidor. A resposta do *back-end* é então interpretada e exibida ao usuário, indicando sucesso ou erro no processo de cadastro.

Figura 70 - Conexão com banco de dados cadastro funcionário



```
// Cadastro de funcionário
app.post('/cadastrarFuncionario', upload.single('foto'), async (req, res) => {
  try {
    const nome = req.body.nome;
    const senha = req.body.senha;
    const email = req.body.email;
    const telefone = req.body.telefone || null;
    const FK_funcao_id = req.body.FK_funcao_id || null;
    let permissoes = req.body['permissoes[]'] || [];

    if (typeof permissoes === "string") {
      permissoes = [permissoes];
    }

    let foto = req.file ? req.file.filename : 'padrao.png';

    if (!nome || !email) return res.status(400).json({ error: 'Campos obrigatórios não preenchidos.' });
    if (telefone && !telefoneValido(telefone)) return res.status(400).json({ error: 'Telefone inválido.' });
    if (req.file) {
      foto = Date.now() + '.jpg';
      const uploadDir = path.join(__dirname, '..', 'uploads', foto);

      await sharp(req.file.path)
        .resize(300, 300)
        .toFormat('jpeg')
        .jpeg({ quality: 90 })
        .toFile(uploadDir);

      fs.unlinkSync(req.file.path); // remove o arquivo original salvo pelo multer
    }

    const checkEmailSql = `SELECT id FROM funcionario WHERE email = ?`;
    connection.query(checkEmailSql, [email], (err, results) => {
      if (err) return res.status(500).json({ error: 'Erro ao verificar e-mail.' });
      if (results.length > 0) return res.status(400).json({ error: 'E-mail já cadastrado.' });

      const senhaFinal = senha && senha.trim() !== "" ? senha : gerarSenhaSegura();

      const sql = `INSERT INTO funcionario (nome, senha, email, foto, telefone, FK_funcao_id)
                   VALUES (?, ?, ?, ?, ?, ?)`;

      connection.query(sql, [nome, senhaFinal, email, foto, telefone, FK_funcao_id], (err, result) => {
        if (err) {
          console.error("Erro no INSERT:", err);
          return res.status(500).json({ error: 'Erro ao cadastrar funcionário' });
        }

        const funcionarioId = result.insertId;

        if (permissoes.length > 0) {
          const values = permissoes.map(p => [p, funcionarioId]);
          const permSql = `INSERT INTO funcionario_permissao (FK_permissao_id, FK_funcionario_id) VALUES ?`;
          connection.query(permSql, [values], (err) => {
            if (err) return res.status(500).json({ error: 'Funcionário criado, mas erro nas permissões.', senhaGerada: senhaFinal });
            return res.status(200).json({ message: 'Funcionário cadastrado com sucesso!', senhaGerada: senhaFinal });
          });
        } else {
          return res.status(200).json({ message: 'Funcionário cadastrado com sucesso!', senhaGerada: senhaFinal });
        }
      });
    });
  } catch (error) {
    console.error("Erro inesperado:", error);
    return res.status(500).json({ error: 'Erro inesperado no servidor.' });
  }
});

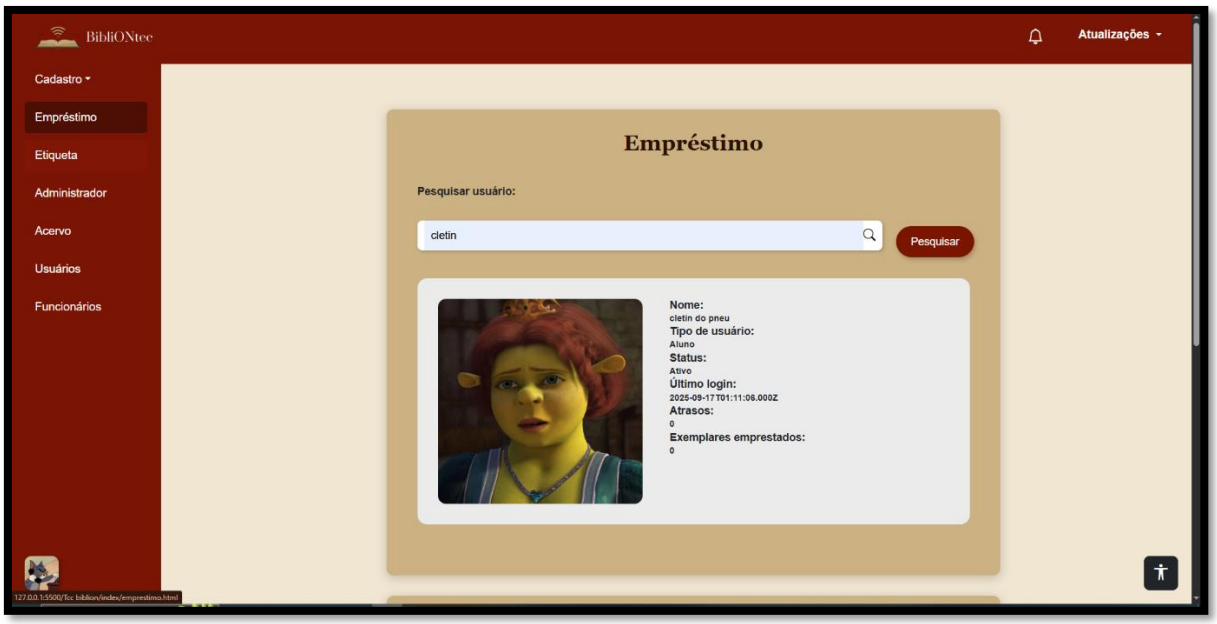

// Função para validar senha forte
function senhaValida(senha) {
  const regex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{8,}$/;
  return regex.test(senha);
}
```

FONTE: AUTORIA PRÓPRIA (2025)

No *back-end* apresentado na figura 70, desenvolvido em Node.js com *Express*, a rota cadastrarFuncionario recebe os dados enviados pelo formulário. O *middleware multer* é utilizado para tratar o *upload* da imagem, enquanto a biblioteca *sharp* realiza o redimensionamento e otimização da foto antes de salvá-la no servidor. Em seguida, o código valida os campos obrigatórios e verifica se o *e-mail* informado já está cadastrado na tabela funcionario. Caso o *e-mail* seja único, é gerada uma senha (caso não tenha sido fornecida) e um comando SQL *INSERT* é executado para armazenar os dados no banco de dados MySQL. Se houver permissões associadas, estas são registradas em uma tabela relacional funcionario_permissao por meio de inserções múltiplas. Ao final, o servidor retorna uma resposta em formato JSON, informando se o cadastro foi realizado com sucesso ou se ocorreu algum erro, assegurando assim a integridade dos dados e a comunicação entre o sistema *web* e o banco.

3.4.7 – Empréstimo

Figura 71 - Tela de empréstimo pesquisar usuário



FONTE: AUTORIA PRÓPRIA (2025)

A figura 71 mostra a tela de pesquisa de usuário no módulo de Empréstimo. Nela, o bibliotecário pode localizar rapidamente um usuário digitando seu nome no campo de busca. Ao realizar a pesquisa, o sistema exibe as informações principais do usuário encontrado, incluindo nome, tipo de usuário (como aluno ou professor), *status*

da conta, último *login* registrado, quantidade de atrasos e número de exemplares atualmente emprestados.

Além disso, é exibida também a foto do usuário, tornando a identificação mais prática e confiável. Essa tela funciona como a etapa inicial do processo de empréstimo, permitindo ao administrador confirmar os dados do usuário antes de prosseguir com a operação.

Figura 72 - Tela empréstimo pesquisar livro



FONTE: AUTORIA PRÓPRIA (2025)

A tela mostrada na figura 72 corresponde à etapa de busca e seleção de exemplares para empréstimo. Após localizar o usuário, o bibliotecário pode pesquisar diretamente pelo título da obra no campo indicado. O sistema retorna os resultados contendo as informações principais do exemplar, como título, autor, editora, a localização onde foi publicado, *status* de disponibilidade e a quantidade de usuários na fila de espera.

Além dos dados textuais, também é exibida a capa do livro, o que facilita a identificação visual da obra. Caso o exemplar esteja disponível, surge o botão “Emprestar”, que permite registrar imediatamente o empréstimo para o usuário selecionado anteriormente.

Há ainda a opção de marcar o filtro “Filtrar livros que estão disponíveis”, garantindo que a busca apresente apenas exemplares que possam ser emprestados naquele momento. Dessa forma, a tela oferece praticidade tanto na consulta quanto na execução do empréstimo, tornando o processo mais rápido e eficiente.

Figura 73 - Tela de empréstimo indisponível



FONTE: AUTORIA PRÓPRIA (2025)

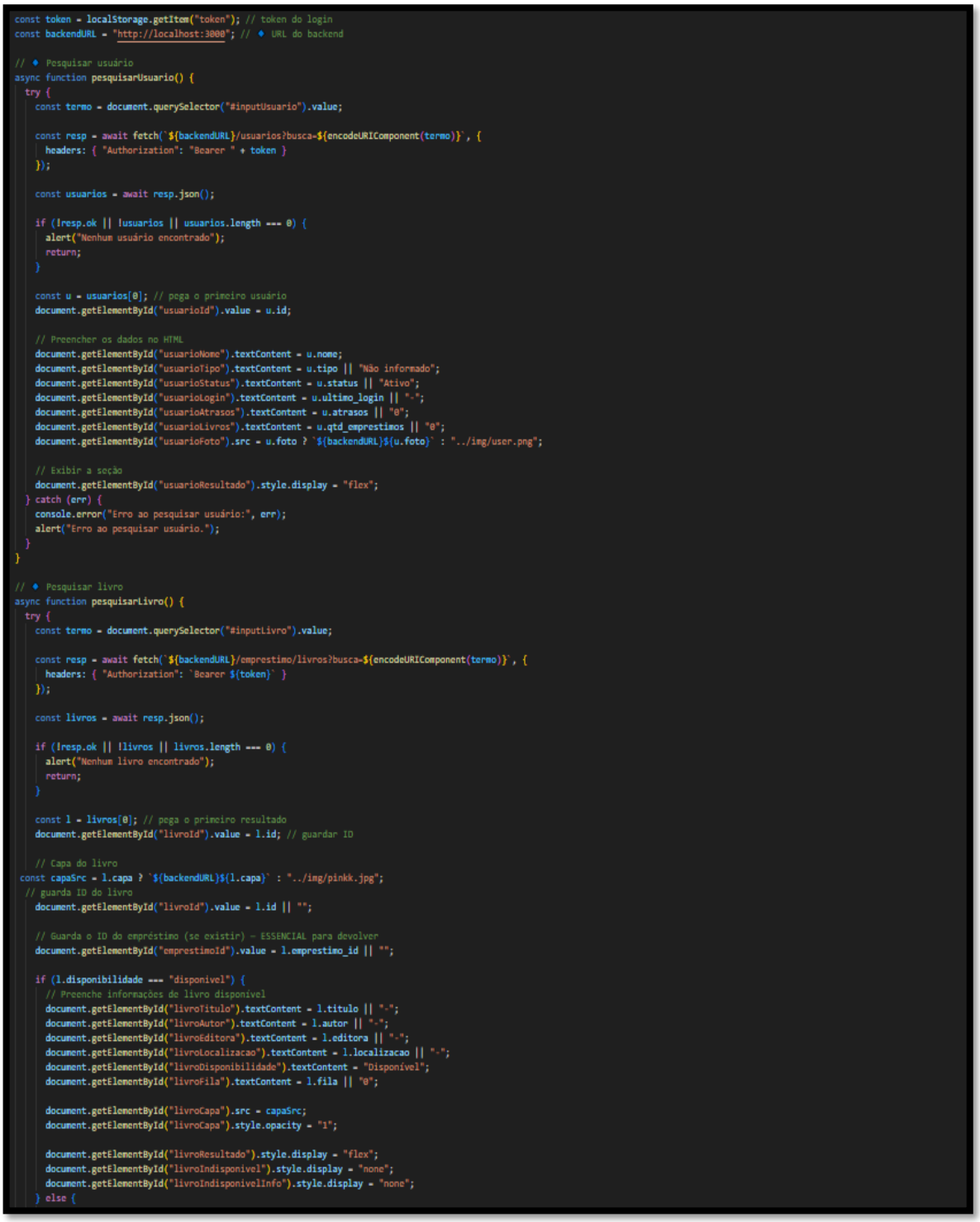
Essa tela mostrada na figura 73 corresponde ao momento em que o sistema identifica que o exemplar pesquisado já se encontra emprestado. Logo no topo, é exibido um aviso em destaque na cor vermelha informando a indisponibilidade do livro.

Além da capa da obra, que está apagada para o efeito visual de indisponibilidade do livro, são mostrados os detalhes do empréstimo atual, como o título do livro, o nome do usuário que está com o exemplar, o *status* (emprestado), a data em que o empréstimo foi realizado e a data prevista para devolução. Também aparece a quantidade de usuários na fila de espera para aquele livro, caso exista.

Na parte inferior, o sistema oferece duas ações possíveis: “Renovar”, que prorroga o prazo de devolução para o mesmo usuário que já está com o exemplar, e

“Devolver”, que registra a devolução do livro, tornando-o novamente disponível para outros usuários.

Figura 74 - Código de empréstimo A



```
const token = localStorage.getItem("token"); // token do login
const backendURL = "http://localhost:3000"; // 🔹 URL do backend

// 🔹 Pesquisar usuário
async function pesquisarUsuario() {
  try {
    const termo = document.querySelector("#inputUsuario").value;

    const resp = await fetch(`${backendURL}/usuarios?busca=${encodeURIComponent(termo)}`, {
      headers: { "Authorization": "Bearer " + token }
    });

    const usuarios = await resp.json();

    if (!resp.ok || !usuarios || usuarios.length === 0) {
      alert("Nenhum usuário encontrado");
      return;
    }

    const u = usuarios[0]; // pega o primeiro usuário
    document.getElementById("usuarioId").value = u.id;

    // Preencher os dados no HTML
    document.getElementById("usuarioNome").textContent = u.nome;
    document.getElementById("usuarioTipo").textContent = u.tipo || "Não informado";
    document.getElementById("usuarioStatus").textContent = u.status || "Ativo";
    document.getElementById("usuarioLogin").textContent = u.ultimo_login || "-";
    document.getElementById("usuarioAtrasos").textContent = u.atrasos || "0";
    document.getElementById("usuarioLivros").textContent = u.qtd_emprestimos || "0";
    document.getElementById("usuarioFoto").src = u.foto ? `${backendURL}${u.foto}` : "../img/user.png";

    // Exibir a seção
    document.getElementById("usuarioResultado").style.display = "flex";
  } catch (err) {
    console.error("Erro ao pesquisar usuário:", err);
    alert("Erro ao pesquisar usuário.");
  }
}

// 🔹 Pesquisar livro
async function pesquisarLivro() {
  try {
    const termo = document.querySelector("#inputLivro").value;

    const resp = await fetch(`${backendURL}/emprestimo/livros?busca=${encodeURIComponent(termo)}`, {
      headers: { "Authorization": `Bearer ${token}` }
    });

    const livros = await resp.json();

    if (!resp.ok || !livros || livros.length === 0) {
      alert("Nenhum livro encontrado");
      return;
    }

    const l = livros[0]; // pega o primeiro resultado
    document.getElementById("livroId").value = l.id; // guardar ID

    // Capa do livro
  const capaSrc = l.capa ? `${backendURL}${l.capa}` : "../img/pinkk.jpg";
   // guarda ID do livro
    document.getElementById("livroId").value = l.id || "";

    // Guarda o ID do empréstimo (se existir) — ESSENCIAL para devolver
    document.getElementById("emprestimoId").value = l.emprestimo_id || "";

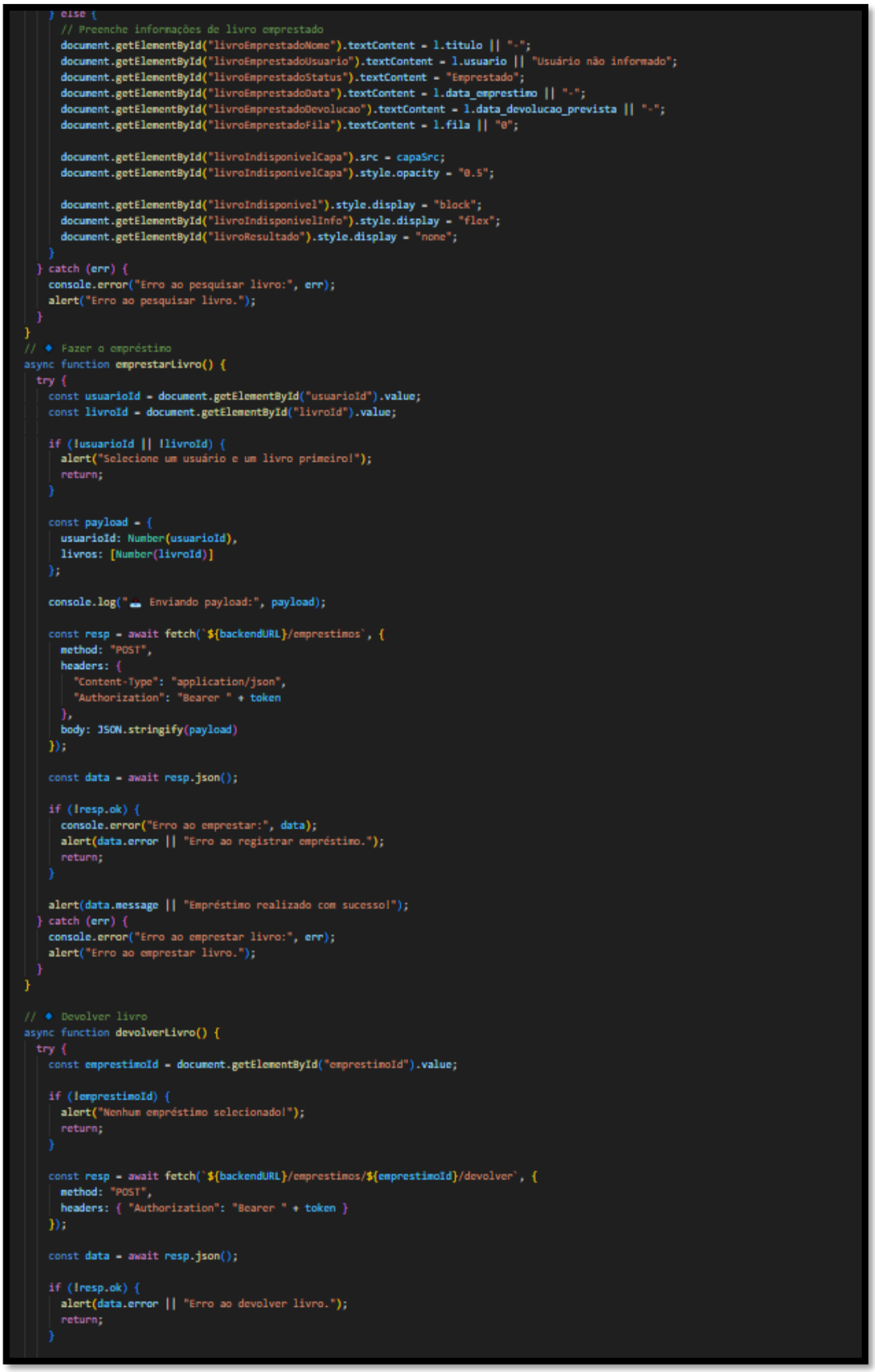
    if (l.disponibilidade === "disponivel") {
      // Preenche informações do livro disponível
      document.getElementById("livroTitulo").textContent = l.titulo || "-";
      document.getElementById("livroAutor").textContent = l.autor || "-";
      document.getElementById("livroEditora").textContent = l.editora || "-";
      document.getElementById("livroLocalizacao").textContent = l.localizacao || "-";
      document.getElementById("livroDisponibilidade").textContent = "Disponível";
      document.getElementById("livroFila").textContent = l.fila || "0";

      document.getElementById("livroCapa").src = capaSrc;
      document.getElementById("livroCapa").style.opacity = "1";

      document.getElementById("livroResultado").style.display = "flex";
      document.getElementById("livroIndisponivel").style.display = "none";
      document.getElementById("livroIndisponivelInfo").style.display = "none";
    } else {
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 75 - Código de empréstimo B



```
    } else {
      // Preenche informações do livro emprestado
      document.getElementById("livroEmprestadoNome").textContent = l.titulo || "-";
      document.getElementById("livroEmprestadoUsuario").textContent = l.usuario || "Usuário não informado";
      document.getElementById("livroEmprestadoStatus").textContent = "Emprestado";
      document.getElementById("livroEmprestadoData").textContent = l.data_emprestimo || "-";
      document.getElementById("livroEmprestadoDevolucao").textContent = l.data_devolucao_prevista || "-";
      document.getElementById("livroEmprestadoFila").textContent = l.fila || "0";

      document.getElementById("livroIndisponivelCapa").src = capaSrc;
      document.getElementById("livroIndisponivelCapa").style.opacity = "0.5";

      document.getElementById("livroIndisponivel").style.display = "block";
      document.getElementById("livroIndisponivelInfo").style.display = "flex";
      document.getElementById("livroResultado").style.display = "none";
    }
  } catch (err) {
    console.error("Erro ao pesquisar livro:", err);
    alert("Erro ao pesquisar livro.");
  }
}
// 🔹 Fazer o empréstimo
async function emprestarLivro() {
  try {
    const usuarioId = document.getElementById("usuarioId").value;
    const livroId = document.getElementById("livroId").value;

    if (!usuarioId || !livroId) {
      alert("Selecione um usuário e um livro primeiro!");
      return;
    }

    const payload = {
      usuarioId: Number(usuarioId),
      livros: [Number(livroId)]
    };

    console.log("📤 Enviando payload:", payload);

    const resp = await fetch(`${backendURL}/emprestimos`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "Authorization": "Bearer " + token
      },
      body: JSON.stringify(payload)
    });

    const data = await resp.json();

    if (!resp.ok) {
      console.error("Erro ao emprestar:", data);
      alert(data.error || "Erro ao registrar empréstimo.");
      return;
    }

    alert(data.message || "Empréstimo realizado com sucesso!");
  } catch (err) {
    console.error("Erro ao emprestar livro:", err);
    alert("Erro ao emprestar livro.");
  }
}

// 🔹 Devolver livro
async function devolverLivro() {
  try {
    const emprestimoId = document.getElementById("emprestimoId").value;

    if (!emprestimoId) {
      alert("Nenhum empréstimo selecionado!");
      return;
    }

    const resp = await fetch(`${backendURL}/emprestimos/${emprestimoId}/devolver`, {
      method: "POST",
      headers: { "Authorization": "Bearer " + token }
    });

    const data = await resp.json();

    if (!resp.ok) {
      alert(data.error || "Erro ao devolver livro.");
      return;
    }
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 76 - Código de empréstimo C



```
    alert(data.message || "Livro devolvido com sucesso!");
    location.reload();
  } catch (err) {
    console.error("Erro ao devolver livro:", err);
    alert("Erro ao devolver livro.");
  }
}

document.addEventListener("DOMContentLoaded", () => {
  const btnDevolver = document.querySelector(".btn-devolver");
  if (btnDevolver) {
    btnDevolver.addEventListener("click", devolverLivro);
  }
  console.log("emprestimo.js carregado. token:", !!token);
});
```

FONTE: AUTORIA PRÓPRIA (2025)

O código apresentado nas figuras 74, 75 e 76 é responsável por toda a lógica do processo de empréstimos e devoluções no sistema. Ele se divide em quatro partes principais: pesquisa de usuários, pesquisa de livros, registro de empréstimos e devolução de exemplares.

Logo no início, é definido o *token* de autenticação, recuperado do *localStorage*, e a URL do *back-end*, que será utilizada em todas as requisições à API. Esse *token* garante que apenas usuários autenticados possam realizar as operações, mantendo a segurança dos dados e evitando acessos indevidos. Esse detalhe é importante porque impede que ações críticas, como emprestar ou devolver livros, sejam realizadas por pessoas não autorizadas.

A primeira parte trata da pesquisa de usuários. A função pesquisarUsuario() captura o termo digitado no campo de busca e envia uma requisição ao servidor para localizar o usuário. Caso o sistema encontre resultados, os dados retornados (como nome, tipo, *status*, *login* e foto) são exibidos diretamente na tela. Esse processo torna o gerenciamento mais prático e permite que o administrador confirme facilmente a identidade do usuário antes de prosseguir com o empréstimo. Se nenhuma informação for encontrada, o sistema emite um alerta informando que não há registros correspondentes.

A segunda parte é a pesquisa de livros, implementada pela função pesquisarLivro(). Ela funciona de maneira semelhante à busca de usuários: envia o termo digitado à API e retorna as informações do exemplar, como título, autor, editora, localização, *status* de disponibilidade e fila de espera. Se o livro estiver disponível, o botão de empréstimo é exibido; caso contrário, a tela mostra os dados do empréstimo ativo, como nome do usuário que pegou o exemplar, data do empréstimo e data prevista para devolução. Essa lógica permite diferenciar claramente os livros que podem ser emprestados dos que já estão em uso, além de fornecer um controle detalhado da circulação de exemplares dentro da biblioteca.

A terceira parte corresponde ao registro de empréstimos. A função emprestarLivro() captura os dados do usuário e do livro selecionados e envia uma requisição *POST* para a API, contendo o *payload* em formato JSON. Se o servidor confirmar a operação, uma mensagem de sucesso é exibida e o empréstimo passa a ser registrado no banco de dados. Esse fluxo garante que cada ação fique devidamente armazenada, possibilitando consultas posteriores e relatórios mais completos. Caso haja algum erro, o sistema alerta o administrador, permitindo que ele tome medidas corretivas rapidamente.

Por fim, a quarta parte é a devolução de livros, implementada pela função devolverLivro(). Ela identifica o empréstimo ativo e envia uma requisição *POST* para a API, sinalizando que o exemplar foi devolvido. Ao receber a resposta positiva, o sistema exibe uma mensagem de confirmação e recarrega a página para atualizar a lista de exemplares disponíveis. Esse processo reduz falhas humanas e garante que os registros da biblioteca estejam sempre atualizados em tempo real.

Além dessas funções principais, há ainda um evento *DOMContentLoaded*, que adiciona dinamicamente o evento de clique no botão de devolução assim que a página é carregada. Isso garante que as interações só sejam ativadas quando os elementos da interface já estiverem prontos, evitando erros de execução e melhorando a experiência do usuário.

Figura 77 - Conexão com banco de dados empréstimo A



```
//================ Emprestimo ================


// ===== Rota: Pesquisar usuários =====
// ===== Rota: Pesquisar usuários =====
app.get('/usuarios', autenticarToken, (req, res) => {
  const termo = req.query.busca;

  const sql = `
    SELECT
      u.id,
      u.nome,
      u.email,
      u.telefone,
      u.foto,
      tu.tipo AS tipo_usuario,
      u.ativo,
      u.ultimo_login,
      -- 👇 Conta quantos empréstimos ativos o usuário tem
      (
        SELECT COUNT(*)
        FROM emprestimo e
        WHERE e.FK_usuario_id = u.id
          AND e.data_real_devolucao IS NULL
      ) AS qtd_emprestimos
    FROM usuario u
    LEFT JOIN tipo_usuario tu ON u.FK_tipo_usuario_id = tu.id
    WHERE u.nome LIKE ? OR u.email LIKE ?
  `;

  connection.query(sql, [`%${termo}%`, `%${termo}%`], (err, results) => {
    if (err) {
      console.error(err);
      return res.status(500).json({ error: "Erro ao buscar usuários" });
    }

    res.json(results.map(u => ({
      id: u.id,
      nome: u.nome,
      tipo: u.tipo_usuario || "Não informado",
      status: u.ativo ? "Ativo" : "Inativo",
      ultimo_login: u.ultimo_login || "-",
      qtd_emprestimos: u.qtd_emprestimos || 0, // 👈 já vem do COUNT
      foto: u.foto ? `/uploads/${u.foto}` : "/uploads/padrao.png"
    })));
  });
});


app.get('/api/usuarios', autenticarToken, (req, res) => {
  const sql = `
    SELECT u.id, u.nome, u.email, u.telefone,
           u.foto,
           u.ultimo_login,       -- 👈 ADICIONAR
           tu.tipo AS tipo,
           u.FK_tipo_usuario_id,
           c.curso AS nome_curso,
           u.serie
    FROM usuario u
    LEFT JOIN tipo_usuario tu ON u.FK_tipo_usuario_id = tu.id
    LEFT JOIN curso c ON u.curso_id = c.id
    WHERE u.FK_instituicao_id = ?
  `;

  connection.query(sql, [req.user.FK_instituicao_id], (err, results) => {
    if (err) {
      console.error("Erro ao buscar usuários:", err);
      return res.status(500).json({ error: "Erro ao buscar usuários" });
    }
    res.json(results);
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 78 - Conexão com banco de dados empréstimo B

```
// ====================== PESQUISAR LIVROS PARA EMPRÉSTIMO ======================
app.get("/emprestimo/livros", autenticarToken, (req, res) => {
  const busca = `%${req.query.busca || ''}%`;

  const sql = `
    SELECT
      l.id,
      l.titulo,
      l.capa,
      l.isbn,
      l.local_publicacao AS localizacao,
      (SELECT GROUP_CONCAT(a.nome SEPARATOR ', ')
         FROM livro_autor la
         JOIN autor a ON la.FK_autor_id = a.id
        WHERE la.FK_livro_id = l.id) AS autores,
      ed.editora AS editora,
      l.disponivel,

      -- Dados do empréstimo (se o livro estiver emprestado)
      e.id AS emprestimo_id,
      DATE_FORMAT(e.data_emprestimo, '%d/%m/%Y') AS data_emprestimo,
      DATE_FORMAT(e.data_devolucao_prevista, '%d/%m/%Y') AS data_devolucao_prevista,
      u.nome AS usuario
    FROM livro l
    LEFT JOIN editora ed ON ed.id = l.FK_editora_id
    LEFT JOIN emprestimo_livro el ON el.FK_livro_id = l.id
    LEFT JOIN emprestimo e
           ON e.id = el.FK_emprestimo_id
          AND e.data_real_devolucao IS NULL
    LEFT JOIN usuario u ON u.id = e.FK_usuario_id
    WHERE l.titulo LIKE ? OR l.isbn LIKE ? OR u.nome LIKE ?
    ORDER BY e.data_emprestimo DESC
    LIMIT 10;
  `;

  connection.query(sql, [busca, busca, busca], (err, rows) => {
    if (err) {
      console.error("Erro ao buscar livros:", err);
      return res.status(500).json({ error: "Erro ao buscar livros" });
    }

    const out = rows.map(l => ({
      id: l.id,
      titulo: l.titulo,
      capa: l.capa ? `/uploads/${l.capa}` : null,
      isbn: l.isbn,
      localizacao: l.localizacao,
      autor: l.autores || null,
      editora: l.editora || null,
      disponibilidade: l.disponivel == 1 ? "disponivel" : "indisponivel",
      emprestimo_id: l.emprestimo_id || null,
      data_emprestimo: l.data_emprestimo || null,
      data_devolucao_prevista: l.data_devolucao_prevista || null,
      usuario: l.usuario || null
    }));

    res.json(out);
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 79 - Conexão com banco de dados empréstimo C



```
// ===== Rota: Criar empréstimo =====
app.post('/emprestimos', autenticarToken, async (req, res) => {
  const { usuarioId, livros } = req.body;

  if (!usuarioId || !Array.isArray(livros) || livros.length === 0) {
    return res.status(400).json({ error: "Dados inválidos. Informe usuário e ao menos um livro." });
  }

  try {
    // 1. Pega configuração de prazo da instituição do token
    const [config] = await pool.query(
      "SELECT duracao_padrao_emprestimo FROM configuracoes_gerais WHERE FK_instituicao_id = ?",
      [req.user.FK_instituicao_id]
    );
    const dias = config[0]?.duracao_padrao_emprestimo || 7;

    const hoje = new Date();
    const devolucaoPrevista = new Date();
    devolucaoPrevista.setDate(hoje.getDate() + dias);

    // 2. Insere o empréstimo
    const [result] = await pool.query(
      "INSERT INTO emprestimo (data_emprestimo, data_devolucao_prevista, FK_usuario_id, FK_instituicao_id) VALUES (?, ?, ?, ?)",
      [hoje, devolucaoPrevista, usuarioId, req.user.FK_instituicao_id]
    );
    const emprestimoId = result.insertId;

    // 3. Relaciona os livros
    for (const livroId of livros) {
      await pool.query("INSERT INTO emprestimo_livro (FK_emprestimo_id, FK_livro_id) VALUES (?, ?)", [emprestimoId, livroId]);
      await pool.query("UPDATE livro SET disponivel = 0 WHERE id = ?", [livroId]); // marca como indisponível
    }

    res.status(201).json({ message: "Empréstimo registrado com sucesso!", emprestimoId });
  } catch (err) {
    console.error("Erro ao registrar empréstimo:", err);
    res.status(500).json({ error: "Erro ao registrar empréstimo" });
  }
});


// ===== Rota: Devolver livro =====
app.post('/emprestimos/:id/devolver', autenticarToken, (req, res) => {
  const emprestimoId = req.params.id;

  const sql = `
    UPDATE emprestimo
    SET data_real_devolucao = NOW()
    WHERE id = ?`;

  connection.query(sql, [emprestimoId], (err) => {
    if (err) {
      console.error("Erro ao devolver:", err);
      return res.status(500).json({ error: "Erro ao devolver livro" });
    }

    const sqlLivro = `
      UPDATE livro
      SET disponivel = 1
      WHERE id IN (SELECT FK_livro_id FROM emprestimo_livro WHERE FK_emprestimo_id = ?)
    `;

    connection.query(sqlLivro, [emprestimoId], (err2) => {
      if (err2) {
        console.warn("⚠️ Coluna 'disponivel' pode não existir na tabela livro");
      }
      res.json({ message: "Livro devolvido com sucesso!" });
    });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

O código apresentado nas figuras 77, 78 e 79 implementa as rotas de *back-end* responsáveis pelo processo de empréstimos dentro do sistema. Ele se conecta ao banco de dados, processa consultas SQL e retorna as informações para o *front-end* de forma organizada.

A primeira parte corresponde à pesquisa de usuários. Quando um termo de busca é enviado, o sistema consulta a tabela de usuários e traz informações como nome, *e-mail*, telefone, tipo de usuário, *status*, último *login* e quantidade de empréstimos ativos. Essa última informação é obtida por meio de uma subconsulta que conta quantos empréstimos ainda não foram devolvidos. Os resultados são processados e enviados em formato JSON, já formatados para exibir corretamente os dados no sistema. Existe também uma rota complementar que retorna todos os usuários de uma instituição, junto com curso e série, permitindo uma listagem mais ampla.

A segunda parte trata da pesquisa de livros. A rota recebe um termo de busca e procura por título, ISBN ou nome do autor. O SQL retorna dados do exemplar, como título, capa, localização, autores, editora e *status* de disponibilidade. Além disso, caso o livro já esteja emprestado, são retornados também os dados do empréstimo ativo, incluindo o usuário responsável e as datas previstas de devolução. Dessa forma, o sistema consegue distinguir entre exemplares disponíveis e emprestados, exibindo as informações adequadas para cada caso.

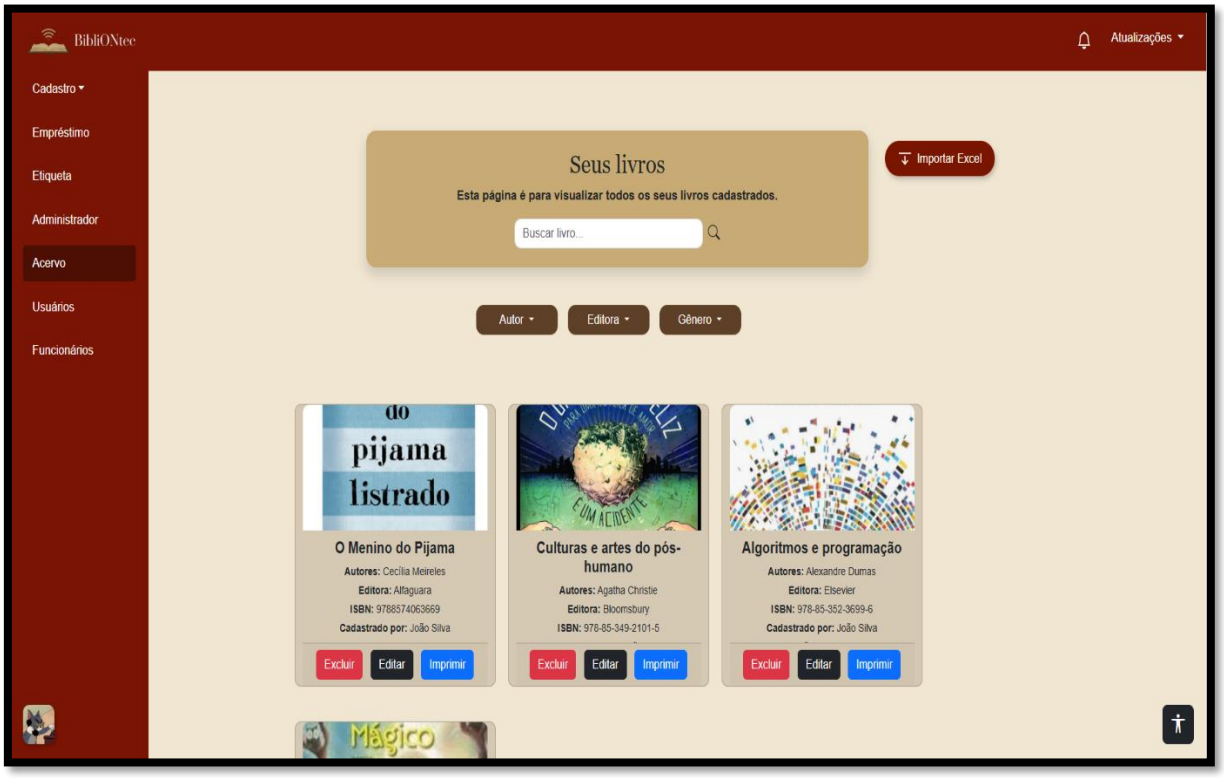
A terceira parte implementa a lógica de criação de empréstimos. Quando o administrador seleciona um usuário e um livro, a rota insere um novo registro na tabela de empréstimos. Antes disso, ela consulta a configuração da instituição para determinar o prazo padrão de devolução. O empréstimo é então gravado no banco, associando o usuário, os livros e a instituição. Em seguida, os exemplares são marcados como “indisponíveis”, impedindo que sejam emprestados novamente até a devolução.

Por fim, temos a rota de devolução de livros. Quando o usuário devolve um exemplar, o sistema atualiza o registro do empréstimo, preenchendo a data de devolução real. Em paralelo, ele altera o *status* do livro para “disponível” na tabela de exemplares, permitindo que volte a ser emprestado. Caso ocorra algum erro, o

sistema retorna mensagens adequadas de falha, mas em caso de sucesso, confirma a devolução para o administrador.

3.4.8 – Acervo de livros

Figura 80 - Tela acervo de livros



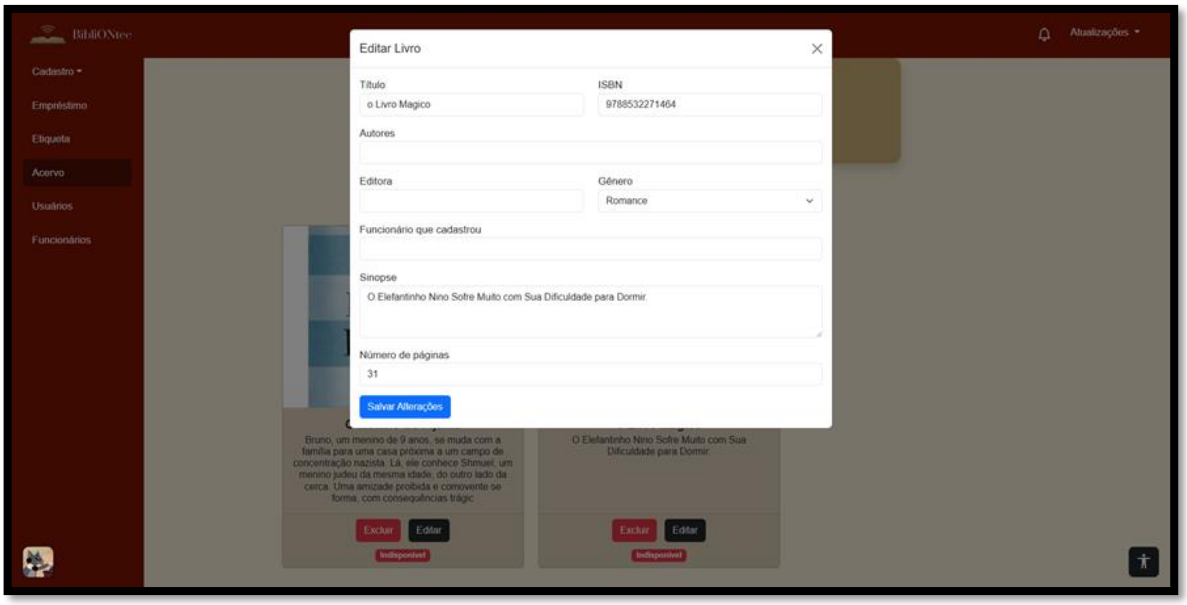
FONTE: AUTORIA PRÓPRIA (2025)

A interface do acervo, como vista na figura 80, é destinada ao bibliotecário e apresenta todos os livros cadastrados no sistema. Cada obra aparece em formato de *card* contendo a capa, título, sinopse resumida e *status* de disponibilidade.

Além disso, existem botões para editar ou excluir livros do acervo. Para que a exclusão seja concluída, é necessária uma confirmação, evitando erros. Na parte superior, há um campo de busca que filtra os livros pelo título, e também botões que permitem organizar a visualização por autor, editora e gênero.

O *layout* é responsivo, ajustando-se bem a diferentes tamanhos de tela, garantindo que a navegação seja confortável tanto em *desktops* quanto em dispositivos móveis.

Figura 81 - Tela *modal* editar livro



FONTE: AUTORIA PRÓPRIA (2025)

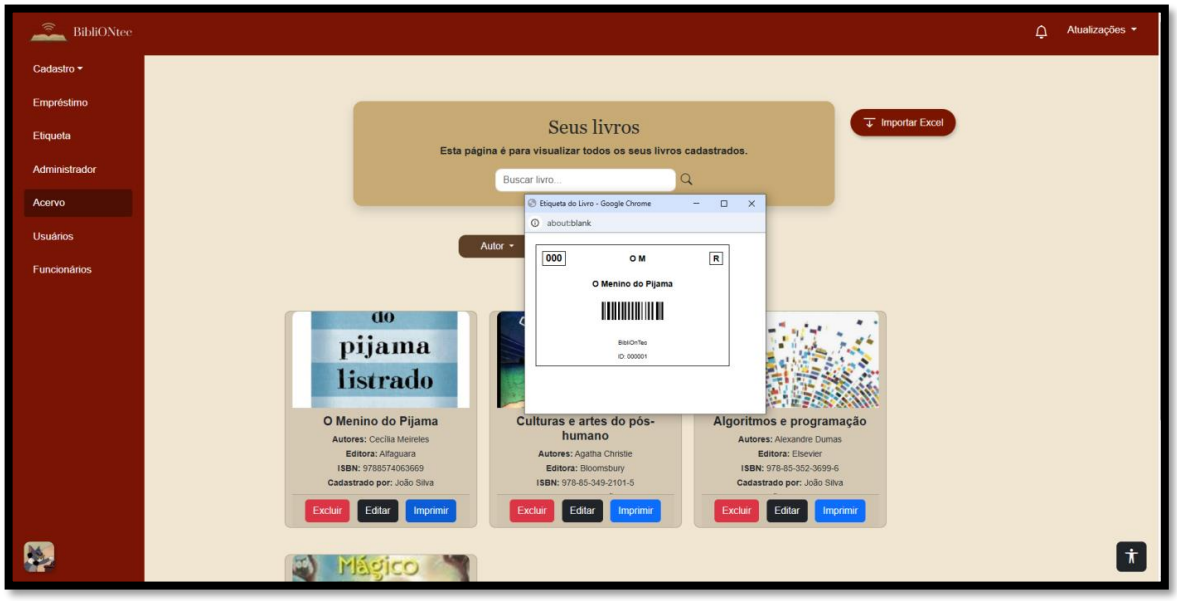
O *modal* (janela sobreposta) de edição, como visto na figura 81, aparece quando o usuário clica no botão Editar de um livro. Ele é exibido como uma janela sobreposta à interface principal, garantindo que o usuário se concentre nas alterações sem perder o contexto do que está editando. No *modal*, os campos do livro são preenchidos automaticamente com os dados cadastrados no banco de dados: Título, ISBN, Autores, Editora, Gênero, Funcionário responsável, Sinopse e Número de páginas.

Esses campos podem ser facilmente alterados conforme necessário, proporcionando flexibilidade ao bibliotecário para manter os dados atualizados. Ao clicar em "Salvar Alterações", as informações modificadas são enviadas ao *back-end*, que atualiza o banco de dados em tempo real, garantindo que a obra refletirá as mudanças de imediato.

Essa funcionalidade evita que o funcionário precise excluir e recadastrar o livro do zero, economizando tempo e evitando duplicações de dados no sistema. Além disso, o *modal* exibe uma mensagem de confirmação logo após as alterações serem salvas, proporcionando ao usuário um *feedback* claro de que os dados foram atualizados corretamente e com sucesso.

Caso o usuário desista das modificações antes de clicar em "Salvar", ele pode fechar o *modal* sem perder os dados ou alterações feitas, o que oferece maior flexibilidade na edição e reduz a chance de erros.

Figura 82 - Tela *modal* etiqueta



FONTE: AUTORIA PRÓPRIA (2025)

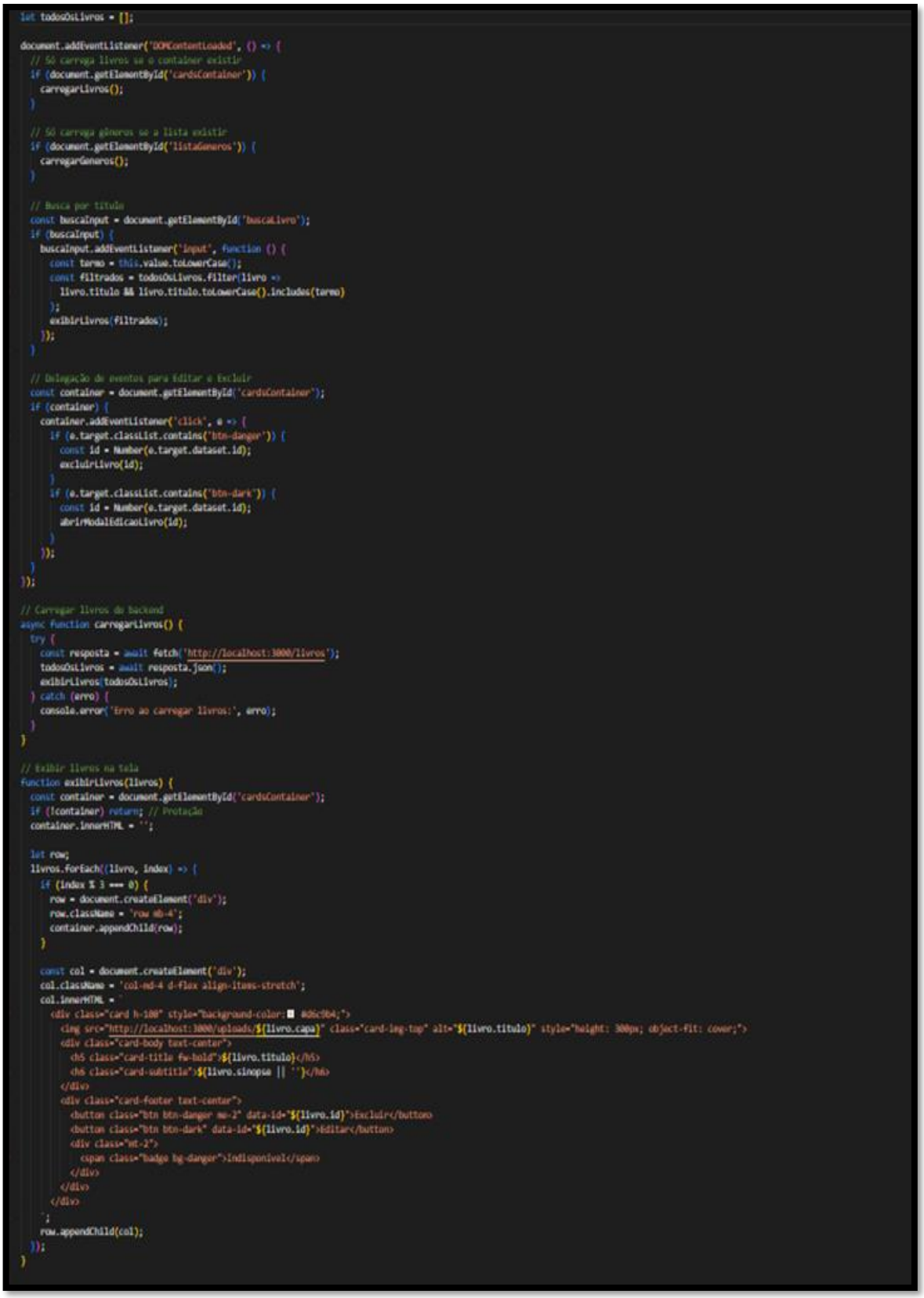
Na figura 82, observa-se a funcionalidade de impressão de etiqueta disponível no módulo de acervo. Após selecionar a opção “Imprimir” em um dos livros cadastrados, o sistema abre uma janela auxiliar que exibe as informações necessárias para identificação da obra.

Essa janela apresenta o título do livro, acompanhado de um código de barras gerado automaticamente pelo sistema, além de um número de registro sequencial. O objetivo é fornecer um rótulo padronizado que pode ser impresso e fixado fisicamente na obra, facilitando os processos de empréstimo e devolução.

Esse recurso garante maior controle do acervo, permitindo que o bibliotecário registre movimentações com rapidez por meio da leitura do código de barras, reduzindo erros e agilizando o atendimento aos usuários.

Além disso, a padronização das etiquetas contribui para a organização interna da biblioteca, já que cada exemplar passa a ter uma identificação clara e única. Essa identificação também auxilia na localização dos livros nas estantes, tornando o processo de busca mais eficiente tanto para os funcionários quanto para os leitores.

Figura 83 - Código acervo de livros A



```
let todosOsLivros = [];

document.addEventListener('DOMContentLoaded', () => {
  // Só carrega livros se o container existir
  if (document.getElementById('cardsContainer')) {
    carregarLivros();
  }

  // Só carrega gêneros se a lista existir
  if (document.getElementById('listaGeneros')) {
    carregarGeneros();
  }

  // Busca por título
  const buscaInput = document.getElementById('buscaLivro');
  if (buscaInput) {
    buscaInput.addEventListener('input', function () {
      const termo = this.value.toLowerCase();
      const filtrados = todosOsLivros.filter(livro =>
        livro.titulo && livro.titulo.toLowerCase().includes(termo)
      );
      exibirLivros(filtrados);
    });
  }

  // Delegação de eventos para Editar e Excluir
  const container = document.getElementById('cardsContainer');
  if (container) {
    container.addEventListener('click', e => {
      if (e.target.classList.contains("btn-danger")) {
        const id = Number(e.target.dataset.id);
        excluirLivro(id);
      }
      if (e.target.classList.contains("btn-dark")) {
        const id = Number(e.target.dataset.id);
        abrirModalEdicaoLivro(id);
      }
    });
  }
});

// Carregar livros do backend
async function carregarLivros() {
  try {
    const resposta = await fetch('http://localhost:3000/livros');
    todosOsLivros = await resposta.json();
    exibirLivros(todosOsLivros);
  } catch (erro) {
    console.error('Erro ao carregar livros:', erro);
  }
}

// Exibir livros na tela
function exibirLivros(livros) {
  const container = document.getElementById('cardsContainer');
  if (!container) return; // Proteção
  container.innerHTML = '';

  let row;
  livros.forEach((livro, index) => {
    if (index % 3 === 0) {
      row = document.createElement('div');
      row.className = 'row mb-4';
      container.appendChild(row);
    }

    const col = document.createElement('div');
    col.className = 'col-md-4 d-flex align-items-stretch';
    col.innerHTML = `
      <div class="card h-100" style="background-color: #d6c9b4;">
        <img src="http://localhost:3000/uploads/${livro.capa}" class="card-img-top" alt="${livro.titulo}" style="height: 300px; object-fit: cover;">
        <div class="card-body text-center">
          <h5 class="card-title fw-bold">${livro.titulo}</h5>
          <h6 class="card-subtitle">${livro.sinopse || ''}</h6>
        </div>
        <div class="card-footer text-center">
          <button class="btn btn-danger me-2" data-id="${livro.id}">Excluir</button>
          <button class="btn btn-dark" data-id="${livro.id}">Editar</button>
          <div class="mt-2">
            <span class="badge bg-danger">Indisponível</span>
          </div>
        </div>
      </div>
    `;
    row.appendChild(col);
  });
}
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 84 - Código acervo de livros B



```
// Carregar gêneros
async function carregarGeneros() {
  const lista = document.getElementById('listaGeneros');
  if (!lista) return; // Proteção

  try {
    const resposta = await fetch('http://localhost:3000/generos');
    const generos = await resposta.json();

    lista.innerHTML = '';

    const itemTodos = document.createElement('li');
    itemTodos.innerHTML = `<a class="dropdown-item item-com-linha" href="#" data-id="0">Todos</a>`;
    lista.appendChild(itemTodos);

    generos.forEach(genero => {
      const item = document.createElement('li');
      item.innerHTML = `<a class="dropdown-item item-com-linha" href="#" data-id="${genero.id}">${genero.genero}</a>`;
      lista.appendChild(item);
    });

    lista.addEventListener('click', e => {
      if (e.target && e.target.matches('a[data-id]')) {
        e.preventDefault();
        const idGenero = parseInt(e.target.dataset.id);
        if (idGenero === 0) {
          exibirLivros(todosOsLivros);
        } else {
          const filtrados = todosOsLivros.filter(livro => livro.FK_genero_id === idGenero);
          exibirLivros(filtrados);
        }
      }
    });
  } catch (erro) {
    console.error('Erro ao carregar gêneros:', erro);
  }
}

// Excluir livro
function excluirLivro(id) {
  if (confirm('Tem certeza que deseja excluir este livro?')) {
    fetch(`http://localhost:3000/livros/${id}`, { method: 'DELETE' })
      .then(res => {
        if (res.ok) {
          carregarLivros();
        } else {
          alert('Erro ao excluir livro');
        }
      })
      .catch(err => console.error('Erro ao excluir livro:', err));
  }
}

// Abrir modal de edição
async function abrirModalEdicaoLivro(id) {
  const livro = todosOsLivros.find(l => l.id === id);
  if (!livro) return;

  // 1️⃣ Carrega todos os gêneros do backend
  let generos = [];
  try {
    const resposta = await fetch('http://localhost:3000/generos'); // ou /generosFiltro
    generos = await resposta.json();
  } catch (erro) {
    console.error('Erro ao carregar gêneros:', erro);
  }

  // 2️⃣ Preenche o select de gêneros com todos os gêneros
  const selectGenero = document.getElementById('genero-livro');
  if (selectGenero) {
    selectGenero.innerHTML = ''; // limpa antes
    generos.forEach(g => {
      const option = document.createElement('option');
      option.value = g.id;
      option.textContent = g.genero;
      // ✅ Marca o gênero do livro como selecionado usando FK
      if (g.id === livro.FK_genero_id) option.selected = true;
      selectGenero.appendChild(option);
    });
  }

  // 3️⃣ Preenche os outros campos do modal
  const campos = [
    'id-livro','titulo-livro','isbn-livro','autores-livro',
    'editora-livro','funcionario-livro','sinopse-livro','paginas-livro'
  ];
  campos.forEach(campo => {
    const el = document.getElementById(campo);
    if (el) {
      if (campo === 'id-livro') el.value = livro.id;
      else el.value = livro[campo.replace('-livro','')] || '';
    }
  });

  // 4️⃣ Abre o modal
  const modalEl = document.getElementById('modal-editar-livro');
  if (modalEl) {
    const modal = new bootstrap.Modal(modalEl);
    modal.show();
  }
}
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 85 - Código acervo de livros C



FONTE: AUTORIA PRÓPRIA (2025)

O código JavaScript, mostrado nas figuras 83, 84 e 85 é responsável pelo acervo de livros tem como objetivo permitir que o usuário visualize, filtre, edite e exclua os registros diretamente pela interface do sistema. Ele atua como uma ponte entre o usuário e o *back-end*, enviando requisições e atualizando a tela conforme as respostas do servidor.

Carregamento inicial:

Assim que a página é aberta, o evento *DOMContentLoaded* é acionado. Nesse momento, o sistema verifica se os elementos da tela existem (como o *container* de livros e a lista de gêneros). Se existirem, chama as funções carregarLivros() e carregarGeneros(). Esse tratamento é importante para evitar erros em páginas que não utilizam esses recursos.

Carregar livros:

A função carregarLivros() realiza uma requisição *fetch* para o *endpoint* http://localhost:3000/livros. O retorno dessa chamada é uma lista em formato JSON com todos os livros cadastrados no banco de dados. Esses dados ficam armazenados no *array* todosOsLivros, e em seguida são enviados para a função exibirLivros(), que gera a interface visual.

Exibir livros:

A função exibirLivros() percorre todos os registros e monta dinamicamente os *cards* na tela, usando *inne*rHTML. Cada *card* contém:

- A imagem da capa do livro.
- O título e a sinopse.
- Botões para excluir ou editar o livro.
- Um selo de disponibilidade (“Indisponível”).

Essa renderização dinâmica garante que qualquer alteração no banco seja refletida automaticamente na interface, sem necessidade de recarregar a página manualmente.

Busca por título:

O campo de busca captura cada tecla digitada pelo usuário. Ele filtra os livros armazenados em todosOsLivros, exibindo apenas os que contêm o termo pesquisado

no título. Isso torna a busca rápida e eficiente, pois não precisa de novas chamadas ao servidor.

Carregar gêneros:

A função carregarGeneros() acessa o *endpoint* http://localhost:3000/generos e preenche a lista de filtros com os gêneros cadastrados. Dessa forma, o usuário pode selecionar, por exemplo, apenas livros do gênero “Romance”. Quando um filtro é clicado, a tela é atualizada exibindo apenas os livros que pertencem ao gênero escolhido.

Excluir livro:

Quando o botão “Excluir” é acionado, o sistema solicita uma confirmação ao usuário. Caso confirmado, é enviado um *fetch* com método *DELETE* para o *endpoint* http://localhost:3000/livros/:id. Se a exclusão for bem-sucedida, a lista de livros é recarregada chamando novamente carregarLivros(). Assim, o livro desaparece da interface sem necessidade de recarregar toda a aplicação.

Editar livro:

O botão “Editar” aciona a função abrir*Modal*EdicaoLivro(id). Essa função localiza os dados do livro no *array* todosOsLivros, faz uma chamada ao *back-end* para carregar todos os gêneros e preencher o campo de seleção do *modal*, preenche automaticamente os campos do formulário (título, ISBN, autores, editora, gênero, funcionário, sinopse e número de páginas) e por fim, abre o *modal* de edição.

Para complementar, o código também oferece a funcionalidade de geração de etiquetas personalizadas para os livros cadastrados. A função gerarConteudoEtiqueta() reúne informações como o número de classificação CDD obtido pela API, as iniciais do título, a letra inicial do gênero e o nome da instituição, compondo um *layout* padronizado. Além disso, o ID (Código de Identificação) do livro é formatado e convertido em código de barras utilizando a biblioteca *JsBarcode*, o que facilita o controle físico do acervo. Esse conteúdo é aberto em uma nova janela e enviado diretamente para impressão, garantindo que cada exemplar possua uma identificação única e de fácil rastreio dentro da biblioteca. Dessa forma, a integração entre sistema e etiquetas físicas reforça a organização do acervo e amplia a eficiência no processo de gestão.

Figura 86 - Conexão com o banco de dados acervo de livros A



```
app.get('/livros/:id', (req, res) => {
  const { id } = req.params;

  const sql = `
  SELECT
    l.id,
    l.titulo,
    l.sinopse,
    l.capa,
    l.paginas,
    l.isbn,
    g.genero,
    e.editora,
    (SELECT GROUP_CONCAT(a.nome SEPARATOR ', ')
     FROM livro_autor la
     JOIN autor a ON la.FK_autor_id = a.id
     WHERE la.FK_livro_id = l.id) AS autores,
    f.nome AS funcionario_cadastrou
  FROM livro l
  LEFT JOIN genero g ON l.FK_genero_id = g.id
  LEFT JOIN editora e ON l.FK_editora_id = e.id
  LEFT JOIN funcionario f ON l.FK_funcionario_id = f.id
  WHERE l.id = ?
  `;

  connection.query(sql, [id], (err, results) => {
    if (err) {
      console.error('Erro ao buscar livro:', err);
      return res.status(500).json({ error: 'Erro no servidor' });
    }

    if (results.length === 0) {
      return res.status(404).json({ error: 'Livro não encontrado' });
    }

    res.json(results[0]); // retorna só o livro encontrado
  });
});

// Atualizar livro com chaves estrangeiras
app.put('/livros/:id', (req, res) => {
  const { id } = req.params;
  const { titulo, isbn, autoresIds, editoraId, generoId, funcionarioId, sinopse, paginas } = req.body;

  const sqlLivro = `
    UPDATE livro
    SET titulo = ?, isbn = ?, FK_editora_id = ?, FK_genero_id = ?, FK_funcionario_id = ?, sinopse = ?, paginas = ?
    WHERE id = ?
  `;

  connection.query(
    sqlLivro,
    [titulo, isbn, editoraId, generoId, funcionarioId, sinopse, paginas, id],
    (err, result) => {
      if (err) {
        console.error('Erro ao atualizar livro:', err);
        return res.status(500).json({ error: 'Erro no servidor' });
      }
      if (result.affectedRows === 0) {
        return res.status(404).json({ error: 'Livro não encontrado' });
      }

      // Atualiza os autores (livro_autor)
      if (Array.isArray(autoresIds)) {
        // Primeiro, deleta os autores antigos
        connection.query('DELETE FROM livro_autor WHERE FK_livro_id = ?', [id], (err) => {
          if (err) {
            console.error('Erro ao deletar autores antigos:', err);
            return res.status(500).json({ error: 'Erro ao atualizar autores' });
          }

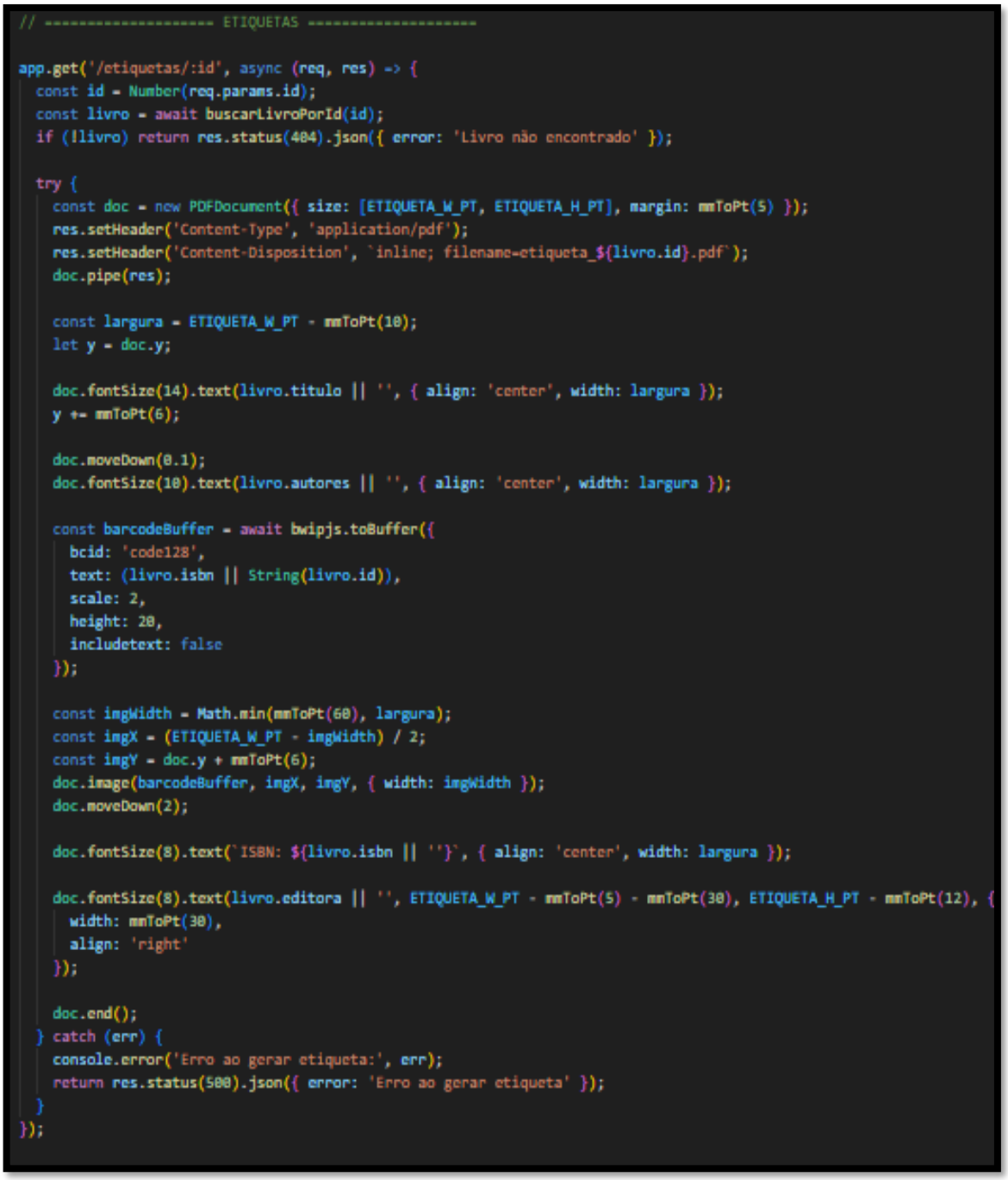
          // Insere os novos autores
          if (autoresIds.length > 0) {
            const valoresAutores = autoresIds.map(autorId => [id, autorId]);
            connection.query('INSERT INTO livro_autor (FK_livro_id, FK_autor_id) VALUES ?', [valoresAutores], (err) => {
              if (err) {
                console.error('Erro ao inserir autores:', err);
                return res.status(500).json({ error: 'Erro ao atualizar autores' });
              }
              return res.json({ message: 'Livro atualizado com sucesso' });
            });
          } else {
            return res.json({ message: 'Livro atualizado com sucesso' });
          }
        });
      } else {
        return res.json({ message: 'Livro atualizado com sucesso' });
      }
    }
  );
});
// Deletar livro com dependências
app.delete('/livros/:id', (req, res) => {
  const { id } = req.params;

  // Primeiro, deletar os autores associados
  connection.query('DELETE FROM livro_autor WHERE FK_livro_id = ?', [id], (err) => {
    if (err) {
      console.error('Erro ao deletar autores do livro:', err);
      return res.status(500).json({ error: 'Erro ao deletar autores do livro' });
    }

    // Depois, deletar o livro
    connection.query('DELETE FROM livro WHERE id = ?', [id], (err, result) => {
      if (err) {
        console.error('Erro ao deletar livro:', err);
        return res.status(500).json({ error: 'Erro ao deletar livro' });
      }
      if (result.affectedRows === 0) {
        return res.status(404).json({ error: 'Livro não encontrado' });
      }
      res.json({ message: 'Livro excluído com sucesso' });
    });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 87 - Conexão com o banco de dados acervo de livros B



```
// ------------------- ETIQUETAS -------------------

app.get('/etiquetas/:id', async (req, res) => {
  const id = Number(req.params.id);
  const livro = await buscarLivroPorId(id);
  if (!livro) return res.status(404).json({ error: 'Livro não encontrado' });

  try {
    const doc = new PDFDocument({ size: [ETIQUETA_W_PT, ETIQUETA_H_PT], margin: mmToPt(5) });
    res.setHeader('Content-Type', 'application/pdf');
    res.setHeader('Content-Disposition', `inline; filename=etiqueta_${livro.id}.pdf`);
    doc.pipe(res);

    const largura = ETIQUETA_W_PT - mmToPt(10);
    let y = doc.y;

    doc.fontSize(14).text(livro.titulo || '', { align: 'center', width: largura });
    y += mmToPt(6);

    doc.moveDown(0.1);
    doc.fontSize(10).text(livro.autores || '', { align: 'center', width: largura });

    const barcodeBuffer = await bwipjs.toBuffer({
      bcid: 'code128',
      text: (livro.isbn || String(livro.id)),
      scale: 2,
      height: 20,
      includetext: false
    });

    const imgWidth = Math.min(mmToPt(60), largura);
    const imgX = (ETIQUETA_W_PT - imgWidth) / 2;
    const imgY = doc.y + mmToPt(6);
    doc.image(barcodeBuffer, imgX, imgY, { width: imgWidth });
    doc.moveDown(2);

    doc.fontSize(8).text(`ISBN: ${livro.isbn || ''}`, { align: 'center', width: largura });

    doc.fontSize(8).text(livro.editora || '', ETIQUETA_W_PT - mmToPt(5) - mmToPt(30), ETIQUETA_H_PT - mmToPt(12), {
      width: mmToPt(30),
      align: 'right'
    });

    doc.end();
  } catch (err) {
    console.error('Erro ao gerar etiqueta:', err);
    return res.status(500).json({ error: 'Erro ao gerar etiqueta' });
  }
});
```

FONTE: AUTORIA PRÓPRIA (2025)

As figuras 86 e 87 mostram o *back-end*, desenvolvido em Node.js integrado ao banco de dados MySQL, estão definidas as rotas responsáveis por manipular os dados dos livros cadastrados no sistema. Essas rotas seguem o padrão *REST*, permitindo que cada operação seja realizada por meio de um tipo de requisição HTTP.

A rota de busca por livro utiliza o método *GET* e recebe o identificador do livro pela URL. Nela, é executada uma consulta SQL que, além dos dados principais da tabela livro, faz junções com outras tabelas do banco para enriquecer o resultado. São buscados o gênero associado, a editora responsável pela publicação, o funcionário que realizou o cadastro e todos os autores vinculados à obra. No caso dos autores, é utilizada a função *GROUP_CONCAT* para reunir todos os nomes em um único campo de texto. Caso o livro não seja encontrado, o sistema retorna uma mensagem de “Livro não encontrado”.

A atualização de livros é feita por meio da rota *PUT*, que recebe os dados enviados pelo *front-end* no corpo da requisição. Primeiramente, a tabela livro é atualizada com as informações principais, (título, ISBN, autores, editora, gênero, funcionário, sinopse e número de páginas) citadas anteriormente. Em seguida, é realizada a atualização da relação entre livros e autores. Para isso, todos os registros anteriores da tabela intermediária livro_autor são removidos, garantindo que não existam vínculos desatualizados, e, logo depois, os novos autores selecionados são inseridos novamente. Essa etapa é essencial porque a relação entre livros e autores é do tipo muitos-para-muitos, exigindo que os dados sejam constantemente sincronizados.

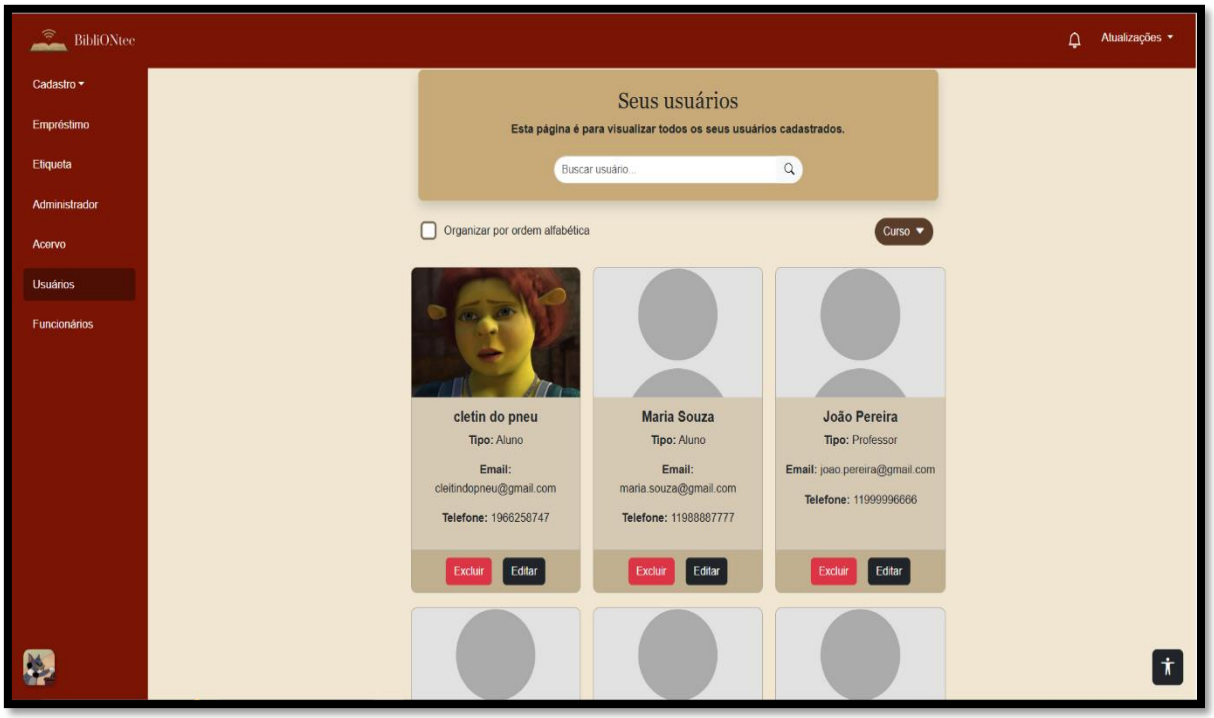
A exclusão de livros, por sua vez, é tratada pela rota *DELETE*. Quando um pedido de exclusão é feito, o sistema primeiramente remove todas as associações existentes na tabela livro_autor, evitando que registros órfãos permaneçam no banco de dados. Somente após essa etapa é que o livro é definitivamente apagado da tabela livro. Caso o identificador fornecido não corresponda a nenhum livro existente, o servidor responde com uma mensagem de erro indicando que o recurso não foi encontrado.

Além disso, o código implementa rotas exclusivas para geração de etiquetas em PDF, tanto individuais quanto múltiplas. Na primeira, acessada via *GET*, é gerada uma etiqueta para um único livro, contendo informações como título, autores e código de barras baseado no ISBN ou ID. Já a segunda, via *POST*, permite selecionar vários livros ao mesmo tempo, montando automaticamente várias etiquetas em formato de grade dentro de uma única página A4. Em ambos os casos, a biblioteca *PDFKit* é utilizada para desenhar os elementos e a *bwip-js* para renderizar os códigos de barras,

garantindo precisão e padronização no resultado final. Essa funcionalidade agrega praticidade ao sistema, pois possibilita a impressão rápida e organizada das etiquetas, otimizando o processo de catalogação e gestão física do acervo.

3.4.9 – Usuários cadastrados

Figura 88 - Tela usuários cadastrados



FONTE: AUTORIA PRÓPRIA (2025)

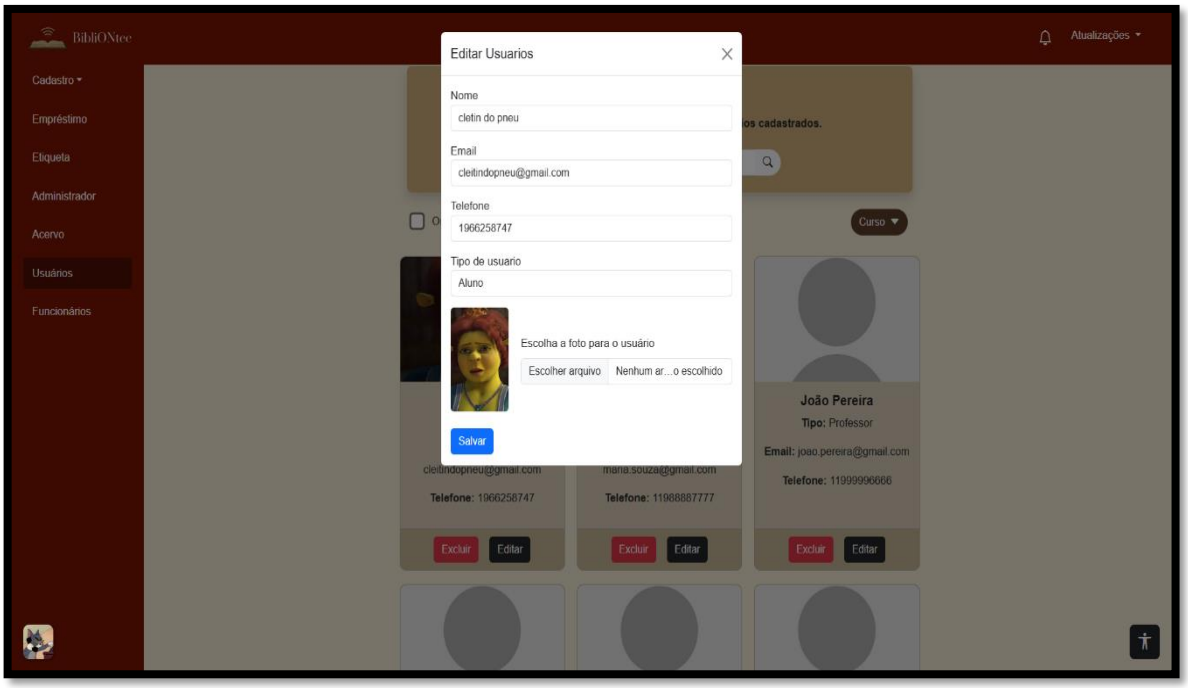
A tela de usuários apresentada na figura 88, mostra de forma organizada todos os usuários cadastrados no sistema.

Cada usuário aparece em um *card* individual, contendo foto, nome, tipo de usuário, *e-mail* e telefone.

Essa estrutura facilita a identificação visual e o gerenciamento, já que cada *card* possui os botões de Editar e Excluir. É necessário uma confirmação ao excluir, para evitar erros.

Além disso, há uma barra de pesquisa no topo, permitindo localizar usuários específicos rapidamente, e uma opção para organizar os resultados em ordem alfabética. Isso torna a navegação prática, especialmente em bibliotecas com muitos cadastros.

Figura 89 - Tela *modal* usuários cadastrados



FONTE: AUTORIA PRÓPRIA (2025)

A figura 89 apresenta o *modal* aberto na tela de usuários cadastrados. Quando o administrador seleciona a opção editar, abre-se um *modal* que exibe os campos de informações do usuário selecionado.

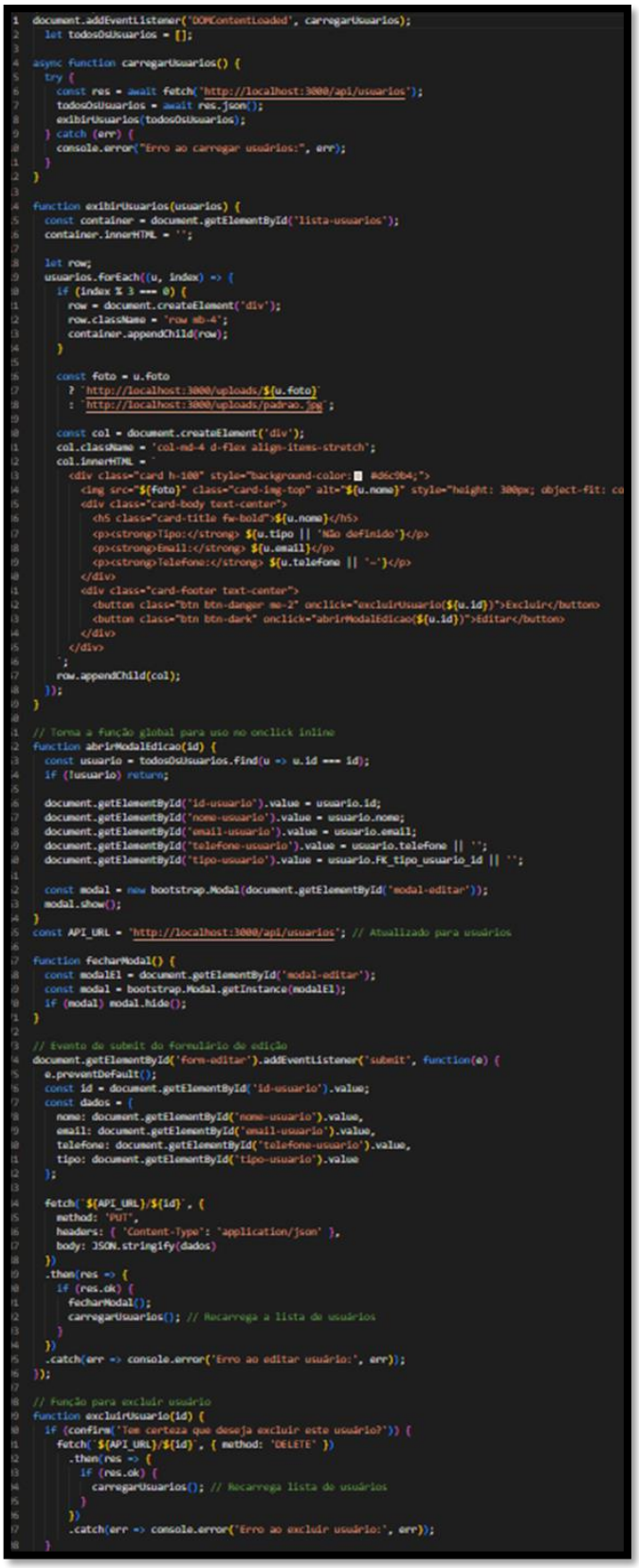
É possível alterar dados como nome, *e-mail*, telefone e função, permitindo que o administrador mantenha as informações sempre atualizadas.

Após a atualização das informações, o botão Salvar envia os novos dados para o banco de dados, garantindo que as alterações fiquem registradas no sistema. O *modal* também exibe uma mensagem de confirmação após a alteração ser salva, assegurando ao administrador que os dados foram modificados corretamente.

Essa funcionalidade é essencial para manter os cadastros atualizados, já que usuários podem mudar de contato ou função ao longo do tempo.

Além disso, a capacidade de editar rapidamente as informações dos usuários contribui para uma gestão mais eficiente e menos suscetível a erros ou dados desatualizados.

Figura 90 - Código usuários cadastrados



```
document.addEventListener('DOMContentLoaded', carregarUsuarios);
  let todosOsUsuarios = [];

async function carregarUsuarios() {
  try {
    const res = await fetch('http://localhost:3000/api/usuarios');
    todosOsUsuarios = await res.json();
    exibirUsuarios(todosOsUsuarios);
  } catch (err) {
    console.error("Erro ao carregar usuários:", err);
  }
}

function exibirUsuarios(usuarios) {
  const container = document.getElementById('lista-usuarios');
  container.innerHTML = '';

  let row;
  usuarios.forEach((u, index) => {
    if (index % 3 === 0) {
      row = document.createElement('div');
      row.className = 'row mb-4';
      container.appendChild(row);
    }

    const foto = u.foto
      ? `http://localhost:3000/uploads/${u.foto}`
      : `http://localhost:3000/uploads/padrao.jpg`;

    const col = document.createElement('div');
    col.className = 'col-md-4 d-flex align-items-stretch';
    col.innerHTML = `
      <div class="card h-100" style="background-color: #d6c9b4;">
        <img src="${foto}" class="card-img-top" alt="${u.nome}" style="height: 300px; object-fit: co
        <div class="card-body text-center">
          <h5 class="card-title fw-bold">${u.nome}</h5>
          <p><strong>Tipo:</strong> ${u.tipo || 'Não definido'}</p>
          <p><strong>Email:</strong> ${u.email}</p>
          <p><strong>Telefone:</strong> ${u.telefone || '-'}</p>
        </div>
        <div class="card-footer text-center">
          <button class="btn btn-danger me-2" onclick="excluirUsuario(${u.id})">Excluir</button>
          <button class="btn btn-dark" onclick="abrirModalEdicao(${u.id})">Editar</button>
        </div>
      </div>
    `;
    row.appendChild(col);
  });
}

// Torna a função global para uso no onclick inline
function abrirModalEdicao(id) {
  const usuario = todosOsUsuarios.find(u => u.id === id);
  if (!usuario) return;

  document.getElementById('id-usuario').value = usuario.id;
  document.getElementById('nome-usuario').value = usuario.nome;
  document.getElementById('email-usuario').value = usuario.email;
  document.getElementById('telefone-usuario').value = usuario.telefone || '';
  document.getElementById('tipo-usuario').value = usuario.FK_tipo_usuario_id || '';

  const modal = new bootstrap.Modal(document.getElementById('modal-editar'));
  modal.show();
}
const API_URL = 'http://localhost:3000/api/usuarios'; // Atualizado para usuários

function fecharModal() {
  const modalEl = document.getElementById('modal-editar');
  const modal = bootstrap.Modal.getInstance(modalEl);
  if (modal) modal.hide();
}

// Evento de submit do formulário de edição
document.getElementById('form-editar').addEventListener('submit', function(e) {
  e.preventDefault();
  const id = document.getElementById('id-usuario').value;
  const dados = {
    nome: document.getElementById('nome-usuario').value,
    email: document.getElementById('email-usuario').value,
    telefone: document.getElementById('telefone-usuario').value,
    tipo: document.getElementById('tipo-usuario').value
  };

  fetch(`${API_URL}/${id}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(dados)
  })
  .then(res => {
    if (res.ok) {
      fecharModal();
      carregarUsuarios(); // Recarrega a lista de usuários
    }
  })
  .catch(err => console.error('Erro ao editar usuário:', err));
});

// Função para excluir usuário
function excluirUsuario(id) {
  if (confirm('Tem certeza que deseja excluir este usuário?')) {
    fetch(`${API_URL}/${id}`, { method: 'DELETE' })
      .then(res => {
        if (res.ok) {
          carregarUsuarios(); // Recarrega lista de usuários
        }
      })
      .catch(err => console.error('Erro ao excluir usuário:', err));
  }
```

FONTE: AUTORIA PRÓPRIA (2025)

É possível ver na figura 90 que logo que a página é carregada (*DOMContentLoaded*), a função carregarUsuarios() é executada. Essa função faz uma requisição *GET* para http://localhost:3888/api/usuarios, buscando todos os usuários cadastrados no banco de dados. O resultado vem em JSON e é armazenado no *array* global todosUsuarios, que será utilizado em outras operações (edição, exclusão, etc.).

A função exibirUsuarios(usuarios) percorre todos os usuários recebidos e cria *cards* dinamicamente com HTML. Cada *card* exibe:

Foto: se o usuário tiver foto cadastrada, ela é exibida; caso contrário, aparece uma imagem padrão.

Nome completo

Tipo de usuário (ex.: professor ou aluno)

E-mail e telefone

Botões de ação:

Editar: abre um *modal* com os dados do usuário já preenchidos.

Excluir: remove o usuário após confirmação.

Ao clicar em Editar, a função abrir*Modal*Edicao(id) é chamada. Ela:

Localiza o usuário correspondente dentro do *array* todosUsuarios.

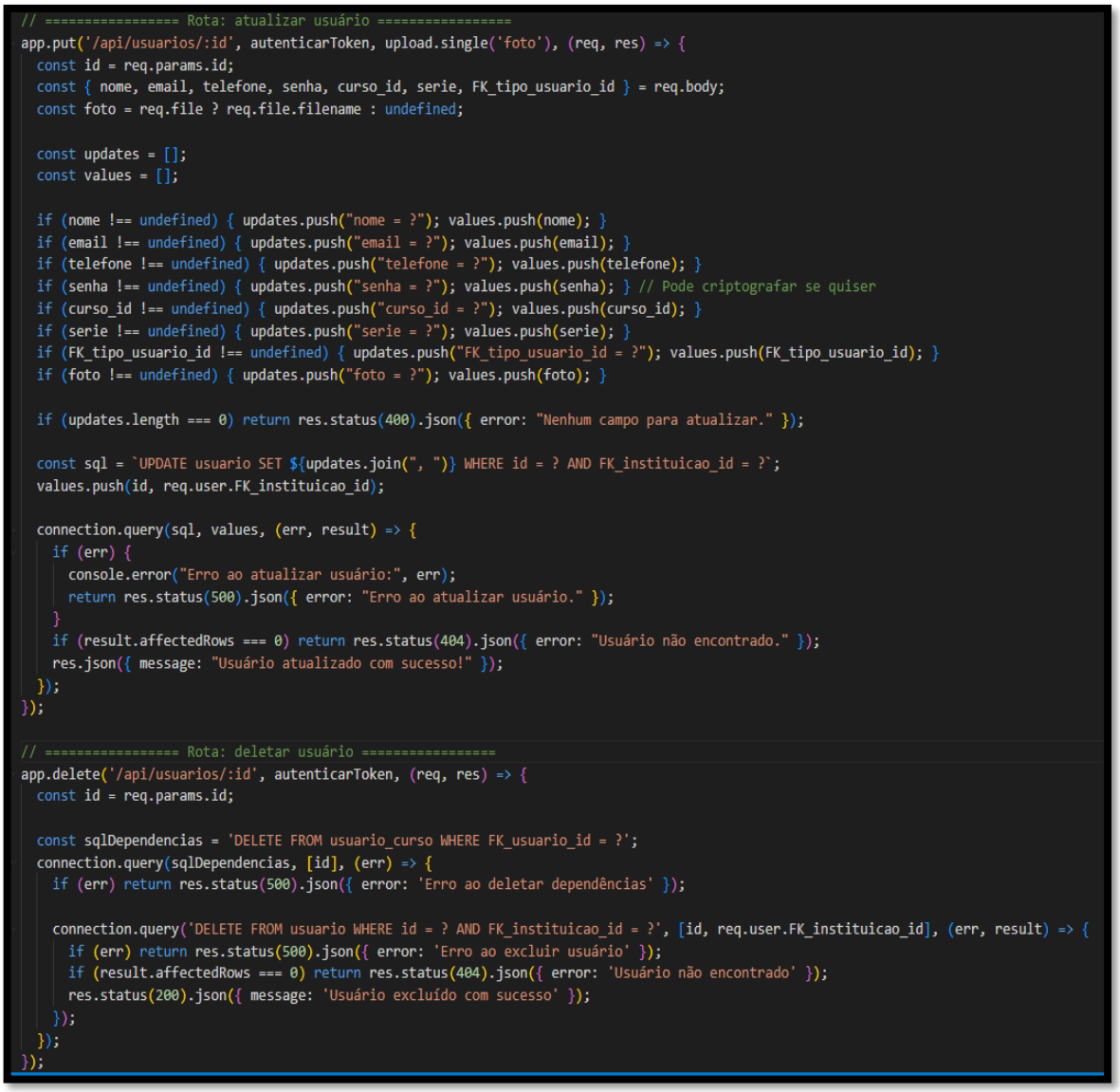
Preenche os campos do formulário dentro do *modal* de edição (nome, *e-mail*, telefone, tipo) com os dados atuais.

Abre o *modal* utilizando o Bootstrap.

Quando o formulário é enviado, os dados são capturados e enviados via requisição *PUT* para http://localhost:3888/api/usuarios/:id. Se a atualização for concluída com sucesso, o *modal* é fechado e a lista de usuários é recarregada na tela.

A função excluirUsuario(id) pede confirmação do administrador. Se confirmado, é feita uma requisição *DELETE* para http://localhost:3888/api/usuarios/:id. Caso o servidor confirme a exclusão, a lista é recarregada, removendo o usuário da interface.

Figura 91 - Conexão com o banco de dados atualizar dados



```
// ================ Rota: atualizar usuário ================
app.put('/api/usuarios/:id', autenticarToken, upload.single('foto'), (req, res) => {
  const id = req.params.id;
  const { nome, email, telefone, senha, curso_id, serie, FK_tipo_usuario_id } = req.body;
  const foto = req.file ? req.file.filename : undefined;

  const updates = [];
  const values = [];

  if (nome !== undefined) { updates.push("nome = ?"); values.push(nome); }
  if (email !== undefined) { updates.push("email = ?"); values.push(email); }
  if (telefone !== undefined) { updates.push("telefone = ?"); values.push(telefone); }
  if (senha !== undefined) { updates.push("senha = ?"); values.push(senha); } // Pode criptografar se quiser
  if (curso_id !== undefined) { updates.push("curso_id = ?"); values.push(curso_id); }
  if (serie !== undefined) { updates.push("serie = ?"); values.push(serie); }
  if (FK_tipo_usuario_id !== undefined) { updates.push("FK_tipo_usuario_id = ?"); values.push(FK_tipo_usuario_id); }
  if (foto !== undefined) { updates.push("foto = ?"); values.push(foto); }

  if (updates.length === 0) return res.status(400).json({ error: "Nenhum campo para atualizar." });

  const sql = `UPDATE usuario SET ${updates.join(", ")} WHERE id = ? AND FK_instituicao_id = ?`;
  values.push(id, req.user.FK_instituicao_id);

  connection.query(sql, values, (err, result) => {
    if (err) {
      console.error("Erro ao atualizar usuário:", err);
      return res.status(500).json({ error: "Erro ao atualizar usuário." });
    }
    if (result.affectedRows === 0) return res.status(404).json({ error: "Usuário não encontrado." });
    res.json({ message: "Usuário atualizado com sucesso!" });
  });
});

// ================ Rota: deletar usuário ================
app.delete('/api/usuarios/:id', autenticarToken, (req, res) => {
  const id = req.params.id;

  const sqlDependencias = 'DELETE FROM usuario_curso WHERE FK_usuario_id = ?';
  connection.query(sqlDependencias, [id], (err) => {
    if (err) return res.status(500).json({ error: 'Erro ao deletar dependências' });

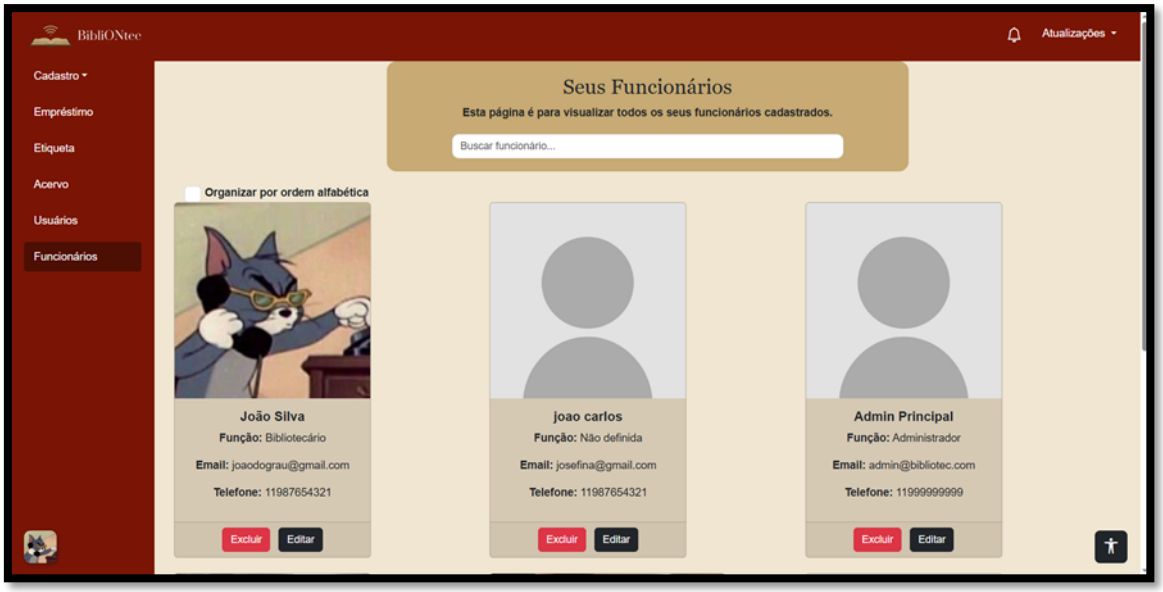
    connection.query('DELETE FROM usuario WHERE id = ? AND FK_instituicao_id = ?', [id, req.user.FK_instituicao_id], (err, result) => {
      if (err) return res.status(500).json({ error: 'Erro ao excluir usuário' });
      if (result.affectedRows === 0) return res.status(404).json({ error: 'Usuário não encontrado' });
      res.status(200).json({ message: 'Usuário excluído com sucesso' });
    });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

É possível ver na figura 91 que o *back-end* implementa duas rotas principais voltadas para a manutenção dos registros de usuários. A primeira, do tipo *PUT* em /api/usuarios/:id, permite atualizar dados como nome, *e-mail*, telefone, senha, curso, série, tipo e até mesmo a foto, montando dinamicamente a *query* SQL apenas com os campos enviados no corpo da requisição. Já a segunda, do tipo *DELETE* em /api/usuarios/:id, é responsável por excluir o usuário, removendo primeiro suas dependências em tabelas associativas antes de apagá-lo definitivamente da tabela principal.

3.4.10 – Funcionários cadastrados

Figura 92 - Tela funcionários cadastrados



FONTE: AUTORIA PRÓPRIA (2025)

A figura 92 mostra a tela de gerenciamento de funcionários, que tem como finalidade listar todos os colaboradores cadastrados no sistema. No topo, há um campo de busca que permite localizar rapidamente um funcionário específico, além da opção de organizar os registros em ordem alfabética, o que facilita a navegação. O *modal* de edição segue o mesmo padrão do visto nos usuários.

Os funcionários são exibidos em *cards* individuais, que apresentam foto (ou uma imagem padrão, caso não haja), nome, função, *e-mail* e telefone. Esse formato torna a visualização mais clara e organizada.

Cada *card* conta com dois botões de ação: Editar e Excluir. Ao clicar em Editar, abre-se um *modal* com os dados do funcionário já preenchidos, permitindo a atualização das informações, que são enviadas ao servidor por meio de uma requisição *PUT*. Já o botão Excluir solicita confirmação antes de enviar uma requisição *DELETE*, removendo o registro do sistema caso o usuário confirme.

Assim, essa tela garante ao administrador um controle prático e seguro dos funcionários, permitindo visualizar, alterar ou excluir informações de forma integrada ao banco de dados.

Figura 93 - Código funcionários cadastrados A

```
document.addEventListener('DOMContentLoaded', carregarFuncionarios);
const API_URL = 'http://localhost:3000/api/funcionarios';
let todosOsFuncionarios = [];

// Só carrega funcionários se o elemento existir
document.addEventListener('DOMContentLoaded', () => {
  if (document.getElementById('lista-funcionarios')) {
    carregarFuncionarios();
  }
});

async function carregarFuncionarios() {
  try {
    const resposta = await fetch(API_URL);
    todosOsFuncionarios = await resposta.json();
    exibirFuncionarios(todosOsFuncionarios);
  } catch (erro) {
    console.error("Erro ao carregar funcionários:", erro);
  }
}

function exibirFuncionarios(funcionarios) {
  const container = document.getElementById('lista-funcionarios');
  if (!container) return; // Se não existir na página, não faz nada

  container.innerHTML = '';
  let row;

  funcionarios.forEach((f, index) => {
    if (index % 3 === 0) {
      row = document.createElement('div');
      row.className = 'row mb-4';
      container.appendChild(row);
    }

    const foto = f.foto
      ? `http://localhost:3000/uploads/${f.foto}`
      : `http://localhost:3000/uploads/padrao.jpg`;

    const col = document.createElement('div');
    col.className = 'col-md-4 d-flex align-items-stretch';

    col.innerHTML = `
      <div class="card h-100" style="background-color: #d6c9b4;">
        <img src="${foto}" class="card-img-top" alt="${f.nome}" style="height: 300px; object-fit: cover;">
        <div class="card-body text-center">
          <h5 class="card-title fw-bold">${f.nome}</h5>
          <p><strong>Função:</strong> ${f.funcao || 'Não definida'}</p>
          <p><strong>Email:</strong> ${f.email}</p>
          <p><strong>Telefone:</strong> ${f.telefone || '—'}</p>
        </div>
        <div class="card-footer text-center">
          <button class="btn btn-danger me-2" onclick="excluirFuncionario(${f.id})">Excluir</button>
          <button class="btn btn-dark" onclick="abrirModalEdicao(${f.id})">Editar</button>
        </div>
      </div>
    `;

    row.appendChild(col);
  });
}
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 94 - Código funcionários cadastrados B



```
function abrirModalEdicao(id) {
  const funcionario = todosOsFuncionarios.find(f => f.id === id);
  if (!funcionario) return;

  // Só preenche se os campos existirem
  const idField = document.getElementById('id-funcionario');
  if (idField) idField.value = funcionario.id;

  const nomeField = document.getElementById('nome-funcionario');
  if (nomeField) nomeField.value = funcionario.nome;

  const emailField = document lementById('email-funcionario');
  if (emailField) emailField.value = funcionario.email;

  const telefoneField = document.getElementById('telefone-funcionario');
  if (telefoneField) telefoneField.value = funcionario.telefone || '';

  const funcaoField = document.getElementById('funcao-funcionario');
  if (funcaoField) funcaoField.value = funcionario.funcao || '';

  const modalElement = document.getElementById('modal-editar');
  if (modalElement) {
    const modal = new bootstrap.Modal(modalElement);
    modal.show();
  }
}

function fecharModal() {
  const modalElement = document.getElementById('modal-editar');
  if (modalElement) {
    modalElement.style.display = 'none';
  }
}

// Só adiciona evento no formulário se ele existir
const formEditar = document.getElementById('form-editar');
if (formEditar) {
  formEditar.addEventListener('submit', function(e) {
    e.preventDefault();
    const id = document.getElementById('id-funcionario')?.value;
    const dados = {
      nome: document.getElementById('nome-funcionario')?.value,
      email: document.getElementById('email-funcionario')?.value,
      telefone: document.getElementById('telefone-funcionario')?.value,
      funcao: document.getElementById('funcao-funcionario')?.value
    };

    fetch(`${API_URL}/${id}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(dados)
    }).then(() => {
      fecharModal();
      carregarFuncionarios();
    });
  });
}

// Função para excluir funcionário
function excluirFuncionario(id) {
  if (confirm('Tem certeza que deseja excluir este funcionário?')) {
    fetch(`${API_URL}/${id}`, { method: 'DELETE' })
      .then(res => {
        if (res.ok) {
          carregarFuncionarios(); // Recarrega lista
        }
      })
      .catch(err => console.error('Erro ao excluir funcionário:', err));
  }
}
```

FONTE: AUTORIA PRÓPRIA (2025)

As figuras 93 e 94 apresentam o código da tela de gerenciamento de funcionários, uma funcionalidade essencial dentro do sistema, responsável por permitir ao administrador visualizar, editar e excluir registros já cadastrados. Essa tela foi projetada para oferecer praticidade e controle, garantindo que todas as informações relacionadas aos funcionários da biblioteca estejam sempre atualizadas e facilmente acessíveis.

Logo ao ser carregada, a página dispara o evento *DOMContentLoaded*, que aciona automaticamente a função carregarFuncionarios. Essa função estabelece a comunicação com a API do sistema por meio de uma requisição do tipo *GET*, solicitando ao servidor todos os dados referentes aos funcionários cadastrados. A resposta recebida é processada e convertida em formato JSON, sendo em seguida armazenada no *array* denominado todosOsFuncionarios. Essa etapa garante que a aplicação disponha de todos os registros prontos para serem exibidos na interface gráfica.

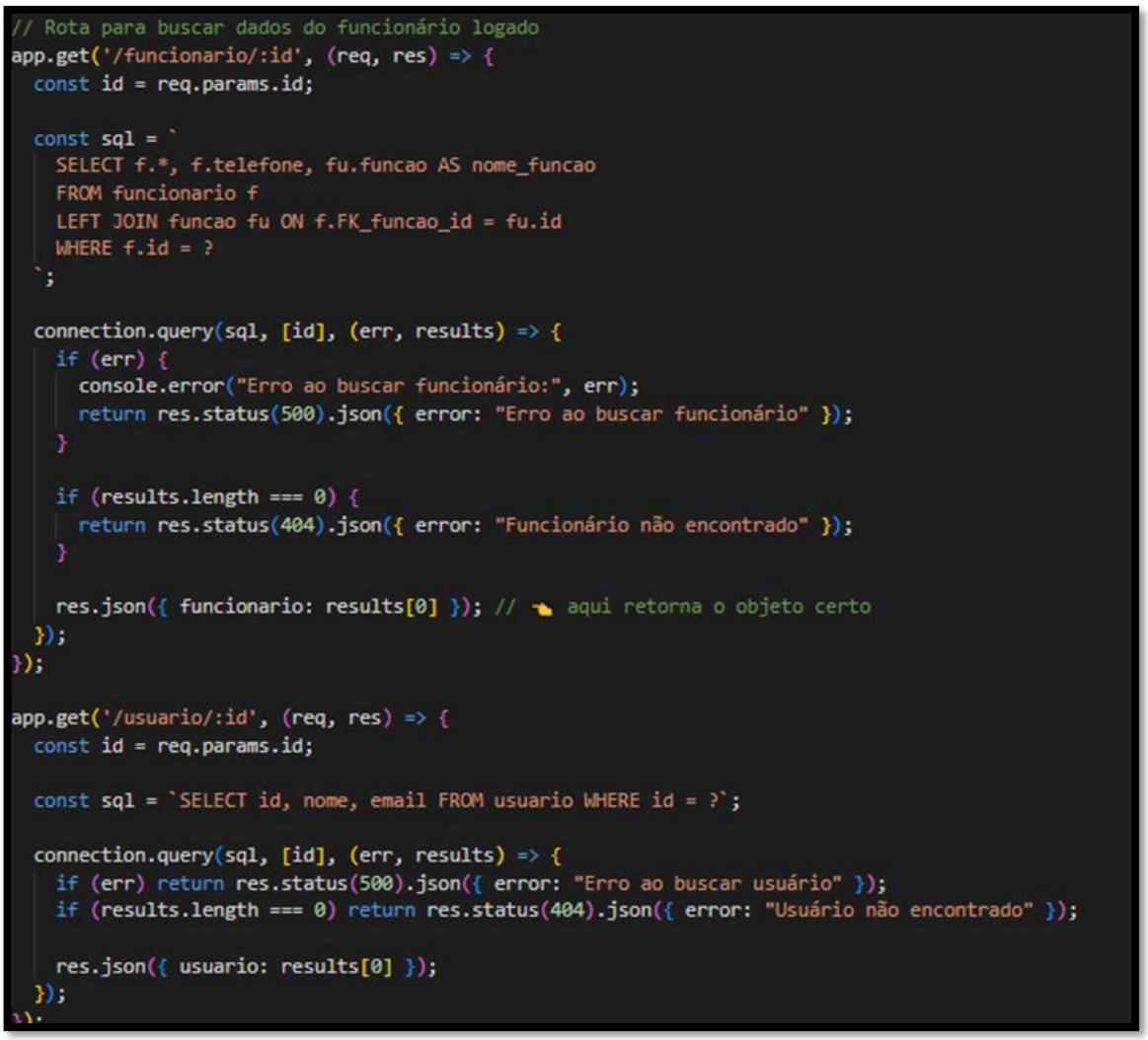
Com os dados já disponíveis, a função exibirFuncionarios é chamada para percorrer o conteúdo armazenado e gerar dinamicamente a exibição dos funcionários na tela. Cada funcionário é representado em um *card* individual, que contém informações essenciais como a fotografia (ou uma imagem padrão, caso o cadastro não possua foto), o nome completo, a função exercida, o endereço de *e-mail* e o número de telefone. Essa forma de apresentação torna a visualização clara e objetiva, permitindo ao administrador identificar rapidamente cada colaborador. Logo abaixo das informações principais, são disponibilizados dois botões de ação: Excluir e Editar, que representam os comandos básicos do gerenciamento de dados.

Ao selecionar a opção Editar, o sistema abre um *modal* (janela sobreposta à tela principal), que já vem automaticamente preenchido com os dados atuais do funcionário selecionado. Esse preenchimento dinâmico evita a necessidade de inserir novamente todas as informações, reduzindo erros e otimizando o tempo de edição. No *modal*, o administrador pode alterar informações como nome, *e-mail*, telefone e função. Assim que as modificações são concluídas e o formulário é submetido, é enviada uma requisição do tipo *PUT* ao servidor. Essa requisição carrega as novas informações que substituem os dados antigos no banco. Caso o servidor confirme que

a atualização foi bem-sucedida, o *modal* é fechado e a lista de funcionários é recarregada automaticamente, refletindo em tempo real as alterações realizadas.

Já ao clicar em Excluir, o sistema implementa uma etapa de segurança solicitando ao administrador uma confirmação da ação, de modo a evitar exclusões acidentais. Apenas após essa confirmação é que o sistema envia uma requisição do tipo *DELETE* para a API, informando qual funcionário deve ser removido. Se o servidor confirmar que a exclusão foi concluída com sucesso, a lista de funcionários na tela é atualizada novamente, garantindo que o registro apagado não esteja mais disponível no sistema, já implementada em todos os “Excluir” do programa.

Figura 95 - Conexão com o banco de dados funcionários cadastrados



```
// Rota para buscar dados do funcionário logado
app.get('/funcionario/:id', (req, res) => {
  const id = req.params.id;

  const sql = `
    SELECT f.*, f.telefone, fu.funcao AS nome_funcao
    FROM funcionario f
    LEFT JOIN funcao fu ON f.FK_funcao_id = fu.id
    WHERE f.id = ?
  `;

  connection.query(sql, [id], (err, results) => {
    if (err) {
      console.error("Erro ao buscar funcionário:", err);
      return res.status(500).json({ error: "Erro ao buscar funcionário" });
    }

    if (results.length === 0) {
      return res.status(404).json({ error: "Funcionário não encontrado" });
    }

    res.json({ funcionario: results[0] }); // 👈 aqui retorna o objeto certo
  });
});

app.get('/usuario/:id', (req, res) => {
  const id = req.params.id;

  const sql = `SELECT id, nome, email FROM usuario WHERE id = ?`;

  connection.query(sql, [id], (err, results) => {
    if (err) return res.status(500).json({ error: "Erro ao buscar usuário" });
    if (results.length === 0) return res.status(404).json({ error: "Usuário não encontrado" });

    res.json({ usuario: results[0] });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

A figura 95 apresenta o *back-end*, desenvolvido em Node.js com MySQL, onde foram criadas rotas específicas para lidar com as informações dos funcionários.

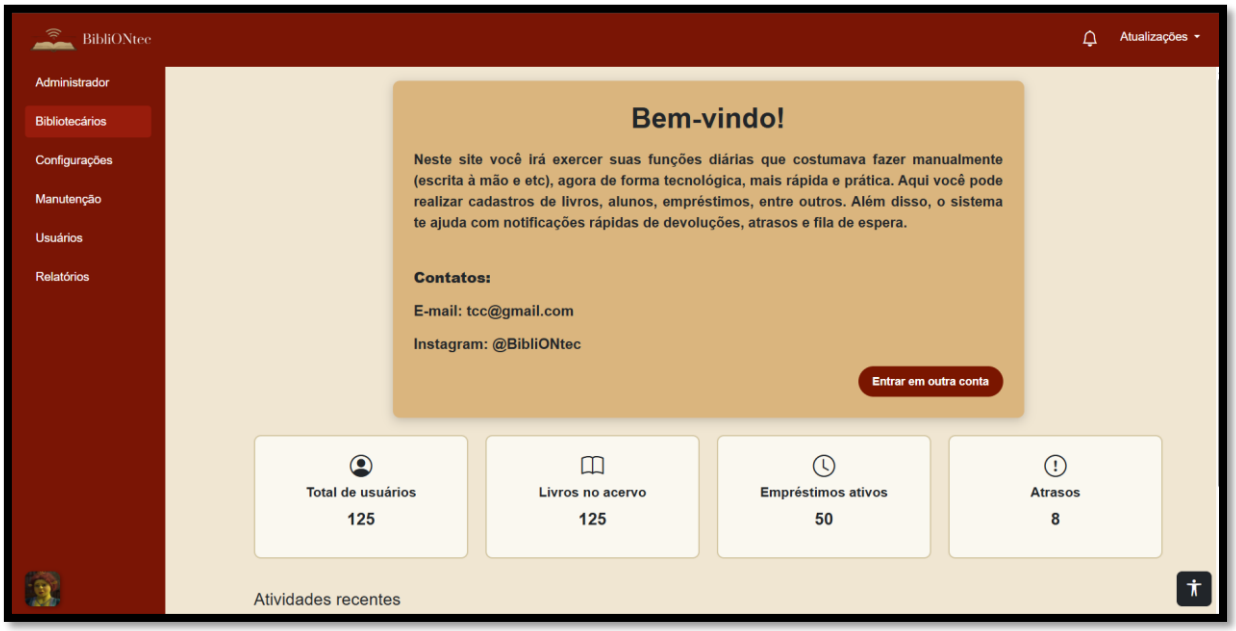
A rota *GET* /funcionario/:id busca os dados de um funcionário específico. A consulta SQL utiliza um JOIN com a tabela funcao para retornar, além das informações básicas (id, nome, *e-mail*, telefone e foto), o nome da função associada ao funcionário. Caso o id não exista, o servidor retorna uma mensagem de erro informando que o funcionário não foi encontrado.

Já a exclusão é realizada pela rota *DELETE* /funcionario/:id. O sistema localiza o registro no banco e, caso ele exista, remove definitivamente o funcionário. Se houver falha na operação ou o registro não for encontrado, uma mensagem de erro é retornada ao *front-end*.

Assim, o *back-end* garante que todas as operações realizadas na tela sejam refletidas corretamente no banco de dados, mantendo a integridade das informações dos funcionários cadastrados.

3.4.11 – Tela Inicial administrador

Figura 96 - Tela Inicial Administrador



FONTE: AUTORIA PRÓPRIA (2025)

A Figura 96 mostra a tela inicial do administrador, exibida logo após o *login* no sistema. O objetivo dela é oferecer uma visão geral e prática das principais funcionalidades disponíveis para a gestão da biblioteca.

Na parte central, há uma mensagem de boas-vindas, explicando que o sistema foi desenvolvido para substituir os processos manuais, tornando tarefas como cadastro de livros, alunos e empréstimos muito mais ágeis. Esse texto também destaca recursos adicionais, como notificações de devoluções, alertas de atraso e gerenciamento de filas de espera, reforçando a eficiência e praticidade que a plataforma proporciona. Além disso, a mensagem funciona como um guia introdutório, facilitando a adaptação de novos administradores ao ambiente digital e evitando dúvidas iniciais sobre o funcionamento do sistema.

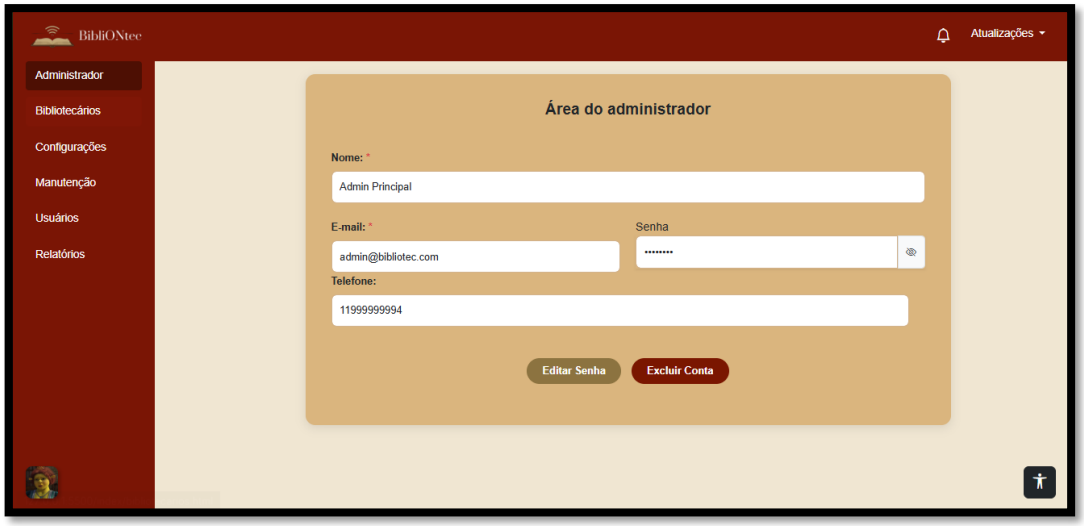
Logo abaixo, o administrador encontra indicadores rápidos organizados em cartões: o total de usuários cadastrados, a quantidade de livros no acervo, o número de empréstimos ativos e os casos de atraso. Esses números funcionam como um painel de estatísticas imediatas, permitindo que o administrador acompanhe o estado geral da biblioteca sem precisar navegar por outras telas. Essa abordagem visual não apenas economiza tempo, mas também possibilita uma tomada de decisão mais assertiva, já que informações relevantes estão sempre visíveis de forma clara e objetiva.

No canto esquerdo, o menu lateral oferece acesso direto às principais seções administrativas, como a gestão de bibliotecários, configurações, manutenção, usuários e relatórios. Isso garante uma navegação simples e intuitiva, centralizando todas as ferramentas de administração em um só lugar. A disposição hierárquica do menu também contribui para reduzir erros, uma vez que cada item está devidamente categorizado, evitando confusão entre funcionalidades semelhantes.

Por fim, o sistema ainda disponibiliza informações de contato, como *e-mail* e Instagram, que podem servir tanto para comunicação institucional quanto para suporte. Essa seção é importante porque estabelece um canal de comunicação entre os usuários e a equipe responsável pelo sistema, reforçando a transparência. Dessa forma, a tela inicial não apenas dá as boas-vindas, mas também cumpre o papel de painel de controle, reunindo de forma clara e organizada os principais dados e recursos que o administrador precisa no dia a dia. Além disso, promove maior autonomia e segurança no gerenciamento da biblioteca, transformando a experiência administrativa em algo moderno, eficiente e confiável.

3.4.12 – Área do administrador

Figura 97 - Tela área administrador



FONTE: AUTORIA PRÓPRIA (2025)

A Figura 97 apresenta a Área do Administrador, cuja finalidade é permitir que o responsável pela gestão do sistema configure e mantenha seus próprios dados de acesso de forma simples e segura.

Na parte central da tela, são exibidos os campos principais do perfil do administrador: nome, *e-mail*, senha e telefone. Esses dados podem ser editados diretamente, garantindo que o responsável pelo sistema mantenha sempre suas informações atualizadas e em conformidade com as necessidades de contato e segurança. O campo de senha conta ainda com um recurso de visualização temporária, possibilitando ao usuário conferir o valor digitado antes de confirmar alterações.

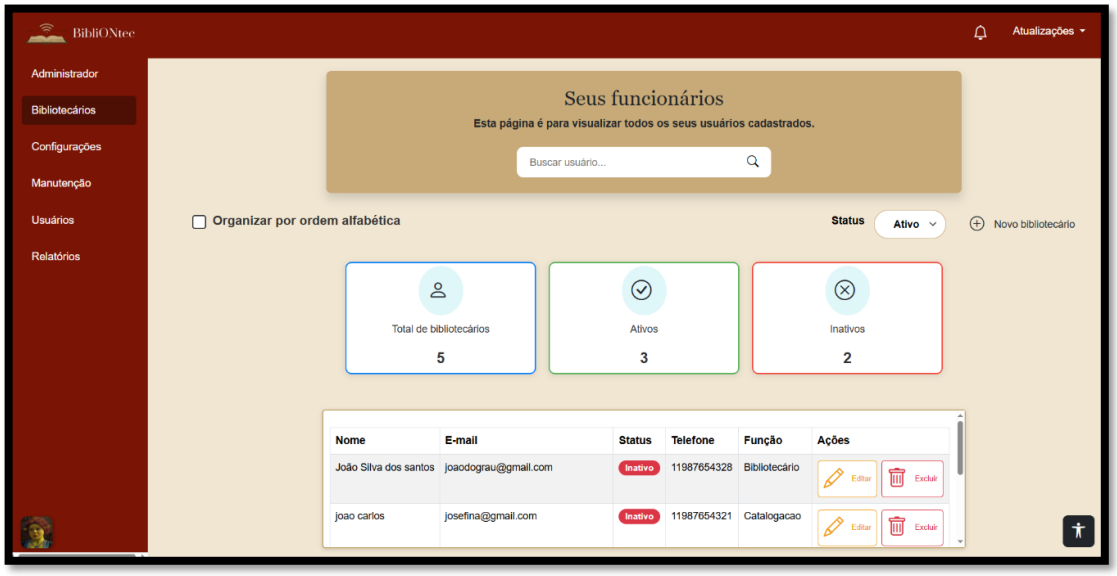
Logo abaixo dos campos, são disponibilizados dois botões principais: “Editar Senha”, que permite atualizar a credencial de acesso de maneira prática, e “Excluir Conta”, que remove permanentemente o perfil do administrador do banco de dados, depois de uma mensagem de confirmação. Essa funcionalidade assegura maior controle sobre os acessos, especialmente em casos de substituição do responsável pela biblioteca.

No lado esquerdo da tela, o menu lateral apresenta as demais seções administrativas, como gestão de bibliotecários, configurações, manutenção, usuários e relatórios. Essa estrutura garante uma navegação clara e organizada, centralizando

as ferramentas de administração em um único espaço. O código *front-end* e *back-end* segue o mesmo padrão visto nas figuras 56, 57 e 58.

3.4.13 – Administração de funcionários

Figura 98 - Tela de Administração dos funcionários



FONTE: AUTORIA PRÓPRIA (2025)

A Figura 98 apresenta a tela de gerenciamento de funcionários, destinada ao administrador do sistema para o controle dos funcionários cadastrados.

Na parte superior, há um campo de busca que permite localizar rapidamente um usuário específico, além de uma opção para organizar por ordem alfabética, facilitando a navegação em bases com muitos registros. Também é possível filtrar os resultados por *status* (ativo ou inativo), garantindo maior precisão na visualização das informações. Ao lado, o botão “Novo bibliotecário” possibilita a inclusão imediata de novos colaboradores ao sistema.

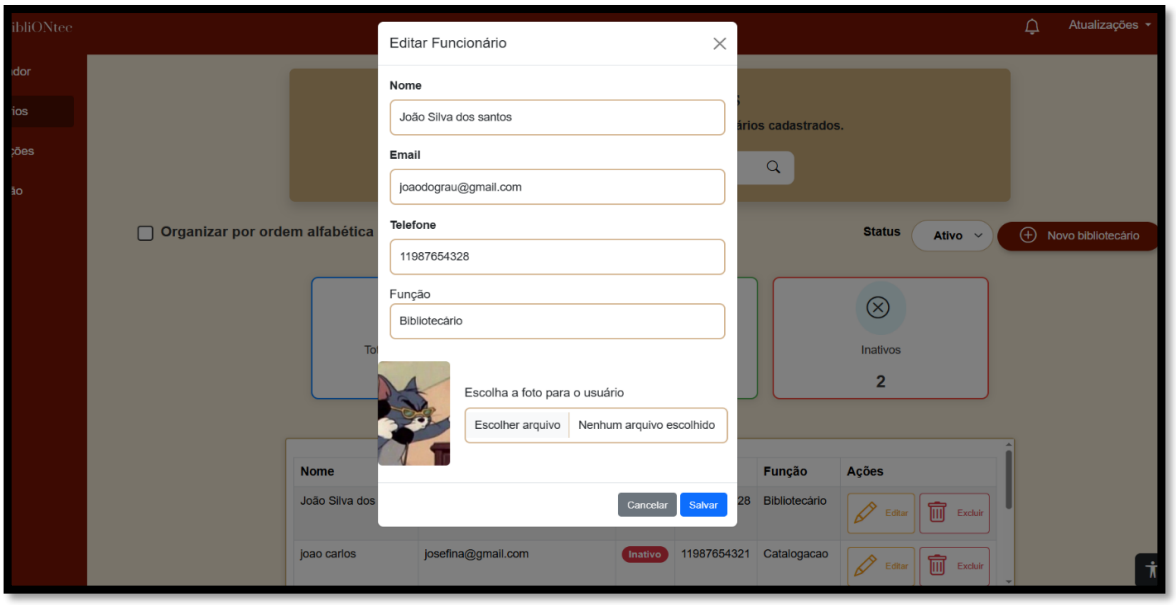
Logo abaixo, são exibidos indicadores rápidos em forma de cartões, apresentando o total de bibliotecários, a quantidade de ativos e os que estão inativos. Esses números funcionam como um painel estatístico, permitindo ao administrador uma visão geral e atualizada do quadro de funcionários da biblioteca.

Na parte inferior, os funcionários cadastrados aparecem em formato de tabela, contendo colunas com nome, *e-mail*, *status*, telefone, função e ações disponíveis. Cada registro conta com dois botões principais: Editar, que abre a possibilidade de atualizar dados do colaborador, e Excluir, que remove permanentemente o cadastro

do sistema após confirmação. Essa organização em tabela garante clareza na visualização e facilidade no gerenciamento das informações.

Dessa forma, a tela de gerenciamento de funcionários oferece ao administrador um controle completo sobre os bibliotecários vinculados à plataforma, reunindo ferramentas de consulta, estatísticas e edição em um único ambiente.

Figura 99 - Tela *modal* de administração editar funcionários



FONTE: AUTORIA PRÓPRIA (2025)

A Figura 99 apresenta a tela de edição de funcionário, exibida em forma de *modal* sobre a interface de gerenciamento de bibliotecários.

Esse *modal* é aberto quando o administrador seleciona a opção Editar na tabela de funcionários. Ele vem preenchido automaticamente com os dados já cadastrados do colaborador, como nome, *e-mail*, telefone e função, permitindo a atualização rápida e prática das informações. Esse preenchimento dinâmico evita a necessidade de digitação redundante e reduz a chance de erros.

Além dos campos de texto, há a possibilidade de alterar a foto do funcionário, por meio do botão de *upload* de imagem. Essa funcionalidade contribui para uma identificação visual mais clara dos colaboradores dentro do sistema, tornando a interface mais personalizada e intuitiva.

Na parte inferior do *modal*, são disponibilizados dois botões principais: Cancelar, que fecha a janela sem aplicar modificações, e Salvar, que dispara uma

requisição ao *back-end* para atualizar os dados no banco de dados. Essa ação é acompanhada por validações, garantindo que apenas informações consistentes sejam persistidas.

Figura 100 - Código Administração de funcionários A



```
// ------------------ EXIBIR EM TABELA ------------------
function exibirFuncionariosTabela(funcionarios) {
  const tbody = document.querySelector("#lista-bibliotecarios tbody");
  if (!tbody) return;

  tbody.innerHTML = "";

  funcionarios.forEach(f => {
    const statusBadge =
      f.status === "Ativo"
        ? '<span class="badge bg-success">Ativo</span>'
        : '<span class="badge bg-danger">Inativo</span>';

    const row = document.createElement("tr");
    row.innerHTML = `
      <td>${f.nome}</td>
      <td>${f.email}</td>
      <td>${statusBadge}</td>
      <td>${f.telefone || "-"}</td>
      <td>${f.funcao || "-"}</td>
      <td>
        <button class="btn btn-outline-warning btn-sm" onclick="abrirModalEdicao(${f.id})"><i class="bi bi-pencil"></i> Editar</button>
        <button class="btn btn-outline-danger btn-sm" onclick="excluirFuncionario(${f.id})"><i class="bi bi-trash"></i> Excluir</button>
      </td>
    `;
    tbody.appendChild(row);
  });
}

// ------------------ ABRIR MODAL DE EDIÇÃO ------------------
async function abrirModalEdicao(id) {
  const funcionario = todosOsFuncionarios.find(f => f.id === id);
  if (!funcionario) return;

  // Carrega todas as funções
  await carregarFuncoes();

  document.getElementById("id-funcionario").value = funcionario.id;
  document.getElementById("nome-funcionario").value = funcionario.nome;
  document.getElementById("email-funcionario").value = funcionario.email;
  document.getElementById("telefone-funcionario").value = funcionario.telefone || "";

  // Preenche o select de funções
  const selectFuncao = document.getElementById("funcao-funcionario");
  if (selectFuncao) {
    selectFuncao.innerHTML = ""; // limpa opções
    todasAsFuncoes.forEach(f => {
      const option = document.createElement("option");
      option.value = f.id;
      option.textContent = f.funcao;
      if (f.id === funcionario.FK_funcao_id) option.selected = true; // marca função atual
      selectFuncao.appendChild(option);
    });
  }

  // Prepara preview da foto
  const preview = document.getElementById("previewBox");
  const fotoAtual = funcionario.foto
    ? `http://localhost:3000/uploads/${funcionario.foto}`
    : `http://localhost:3000/uploads/padrao.jpg`;
  preview.style.backgroundImage = `url('${fotoAtual}')`;

  // Abre modal
  const modal = new bootstrap.Modal(document.getElementById("modal-editar"));
  modal.show();
}
```

FONTE: AUTORIA PRÓPRIA (2025)

Figura 101 - Código Administração de funcionários B



```
// ------------------ SALVAR EDIÇÃO ------------------
const formEditar = document.getElementById("form-editar");
if (formEditar) {
  formEditar.addEventListener("submit", function (e) {
    e.preventDefault();
    const id = document.getElementById("id-funcionario").value;
    const formData = new FormData();
    formData.append("nome", document.getElementById("nome-funcionario").value);
    formData.append("email", document.getElementById("email-funcionario").value);
    formData.append("telefone", document.getElementById("telefone-funcionario").value);
    formData.append("FK_funcao_id", document.getElementById("funcao-funcionario").value);

    const fotoInput = document.getElementById("foto-funcionario");
    if (fotoInput && fotoInput.files.length > 0) {
      formData.append("foto", fotoInput.files[0]);
    }
    fetch(`${API_URL_FUNCIONARIOS}/${id}`, {
      method: "PUT",
      headers: { "Authorization": "Bearer " + localStorage.getItem("token") },
      body: formData
    })
    .then(async res => {
      if (!res.ok) {
        const errData = await res.json().catch(() => ({}));
        throw new Error(errData.error || "Erro ao atualizar funcionário");
      }
      carregarFuncionarios();
      const modalEl = document.getElementById("modal-editar");
      const modal = bootstrap.Modal.getInstance(modalEl);
      modal.hide();
    })
    .catch(err => console.error("Erro ao editar funcionário:", err));
  });
}
// ------------------ EXCLUIR ------------------
function excluirFuncionario(id) {
  if (confirm("Tem certeza que deseja excluir este funcionário?")) {
    fetch(`${API_URL_FUNCIONARIOS}/${id}`, {
      method: "DELETE",
      headers: { "Authorization": "Bearer " + localStorage.getItem("token") }
    })
      .then(res => {
        if (res.ok) carregarFuncionarios();
      })
      .catch(err => console.error("Erro ao excluir funcionário:", err));
  }
}
```

FONTE: AUTORIA PRÓPRIA (2025)

As Figuras 100 e 101 apresentam o código responsável pelo gerenciamento de funcionários no *front-end* do sistema, abrangendo as operações de listagem, edição e exclusão.

Na primeira parte, a função exibirFuncionariosTabela recebe a lista de funcionários carregada do banco de dados e renderiza dinamicamente uma tabela HTML. Para cada funcionário, são criadas colunas que exibem nome, *e-mail*, *status*, telefone e função, além de botões de ação para editar e excluir o registro. O *status* é exibido com cores diferenciadas (verde para ativos e vermelho para inativos), facilitando a identificação visual.

Em seguida, a função abrir*Modal*Edicao é responsável por abrir o *modal* de edição ao clicar no botão correspondente. Os dados do funcionário selecionado são carregados e exibidos nos campos do formulário (nome, *e-mail*, telefone e função). O código também popula o campo de seleção de função com todas as opções disponíveis e destaca a função atual do colaborador. Além disso, prepara a pré-visualização da foto do usuário, exibindo a imagem já cadastrada ou uma foto padrão, caso não exista.

Na terceira parte, o código trata o processo de atualização. O formulário de edição (*form*-editar) captura as alterações feitas e cria um objeto *FormData* para envio ao *back-end*. Os dados, incluindo uma possível nova foto, são enviados em uma requisição HTTP do tipo *PUT*, autenticada com *token* de acesso. Em caso de sucesso, o *modal* é fechado e a lista de funcionários é atualizada automaticamente, garantindo que a interface reflita em tempo real as modificações feitas.

Por fim, a função excluirFuncionario implementa a remoção de registros. Antes da exclusão, o sistema exibe uma mensagem de confirmação, prevenindo exclusões acidentais. Se confirmada, é realizada uma requisição HTTP *DELETE* para a API. Caso a operação seja bem-sucedida, a lista de funcionários é recarregada, mantendo a consistência entre a interface e o banco de dados.

Dessa forma, o conjunto de funções garante ao administrador um controle completo sobre os colaboradores, permitindo visualizar, editar e excluir registros. O código *back-end* segue o mesmo padrão visto na edição e exclusão de funcionários na figura 95.

3.4.14 – Administração de configurações

Figura 102 - Tela configuração de prazos e regras



FONTE: AUTORIA PRÓPRIA (2025)

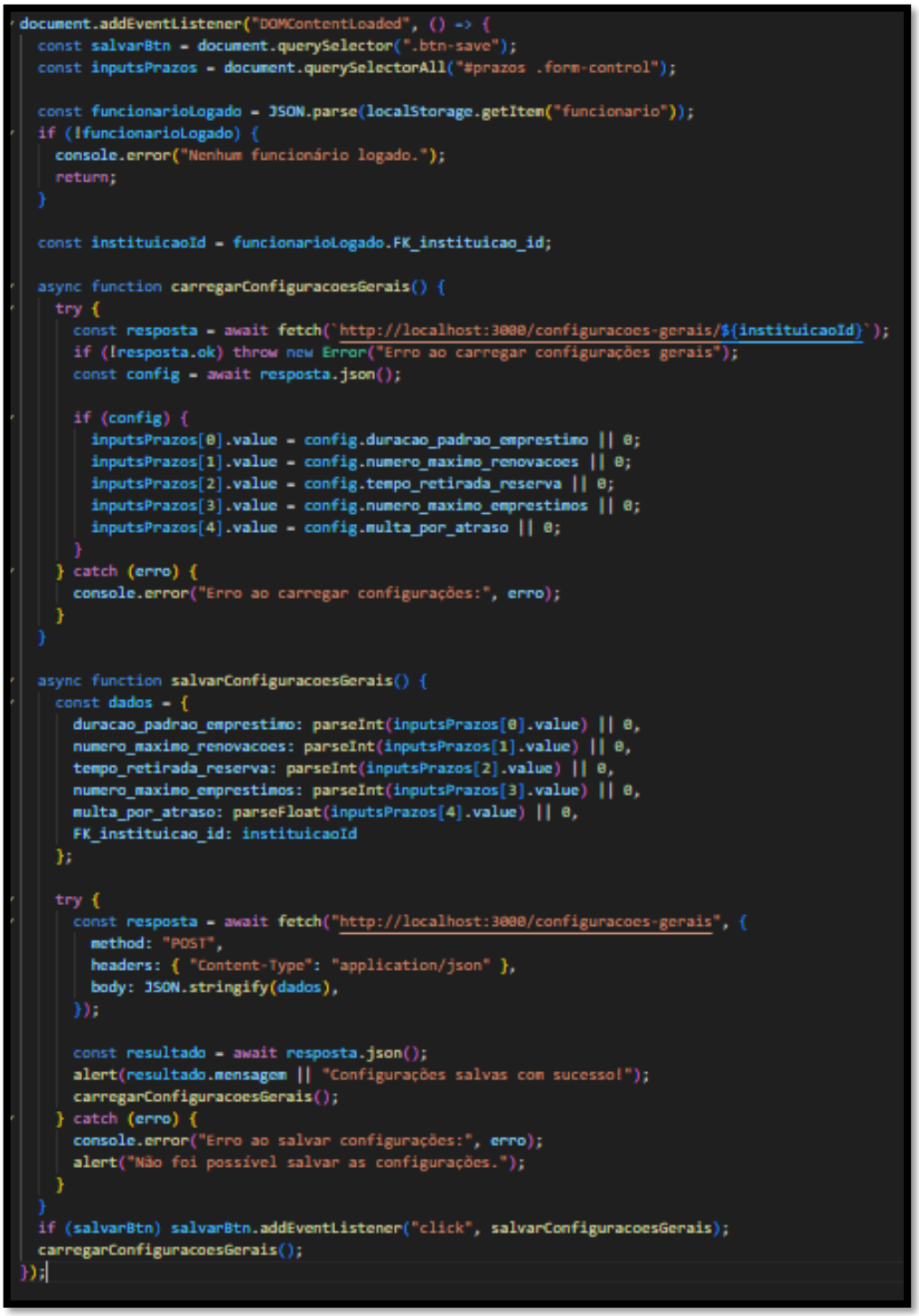
A Figura 102 apresenta a tela de Configurações do Sistema, responsável por definir as regras gerais de funcionamento da biblioteca digital.

Na parte superior, encontram-se abas de navegação que organizam as opções em diferentes categorias: Prazos e regras, Permissões, Dados institucionais e Notificações. Essa estrutura torna o processo de configuração mais intuitivo e segmentado, permitindo que o administrador localize rapidamente o tipo de ajuste desejado.

Na aba exibida, intitulada “Prazos e regras gerais”, são disponibilizados campos para o controle de parâmetros essenciais: duração padrão do empréstimo, tempo para retirada da reserva, número máximo de renovações, quantidade máxima de empréstimos por usuário e o valor da multa por atraso. Esses valores podem ser ajustados de acordo com a política de cada instituição, assegurando flexibilidade na aplicação das normas.

No canto superior direito, o botão “Salvar alterações” permite que os dados informados sejam enviados ao servidor e registrados no banco de dados. Essa funcionalidade garante que todas as modificações tenham efeito imediato sobre o funcionamento do sistema, refletindo nas operações de empréstimo, devolução e reserva.

Figura 103 - Código configuração de prazos e regras



```
document.addEventListener("DOMContentLoaded", () => {
  const salvarBtn = document.querySelector(".btn-save");
  const inputsPrazos = document.querySelectorAll("#prazos .form-control");

  const funcionarioLogado = JSON.parse(localStorage.getItem("funcionario"));
  if (!funcionarioLogado) {
    console.error("Nenhum funcionário logado.");
    return;
  }

  const instituicaoId = funcionarioLogado.FK_instituicao_id;

  async function carregarConfiguracoesGerais() {
    try {
      const resposta = await fetch(`http://localhost:3000/configuracoes-gerais/${instituicaoId}`);
      if (!resposta.ok) throw new Error("Erro ao carregar configurações gerais");
      const config = await resposta.json();

      if (config) {
        inputsPrazos[0].value = config.duracao_padrao_emprestimo || 0;
        inputsPrazos[1].value = config.numero_maximo_renovacoes || 0;
        inputsPrazos[2].value = config.tempo_retirada_reserva || 0;
        inputsPrazos[3].value = config.numero_maximo_emprestimos || 0;
        inputsPrazos[4].value = config.multa_por_atraso || 0;
      }
    } catch (erro) {
      console.error("Erro ao carregar configurações:", erro);
    }
  }

  async function salvarConfiguracoesGerais() {
    const dados = {
      duracao_padrao_emprestimo: parseInt(inputsPrazos[0].value) || 0,
      numero_maximo_renovacoes: parseInt(inputsPrazos[1].value) || 0,
      tempo_retirada_reserva: parseInt(inputsPrazos[2].value) || 0,
      numero_maximo_emprestimos: parseInt(inputsPrazos[3].value) || 0,
      multa_por_atraso: parseFloat(inputsPrazos[4].value) || 0,
      FK_instituicao_id: instituicaoId
    };

    try {
      const resposta = await fetch("http://localhost:3000/configuracoes-gerais", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(dados),
      });

      const resultado = await resposta.json();
      alert(resultado.mensagem || "Configurações salvas com sucesso!");
      carregarConfiguracoesGerais();
    } catch (erro) {
      console.error("Erro ao salvar configurações:", erro);
      alert("Não foi possível salvar as configurações.");
    }
  }
  if (salvarBtn) salvarBtn.addEventListener("click", salvarConfiguracoesGerais);
  carregarConfiguracoesGerais();
});
```

FONTE: AUTORIA PRÓPRIA (2025)

O código apresentado na Figura 103 corresponde à lógica do *front-end* responsável por carregar e salvar as configurações gerais.

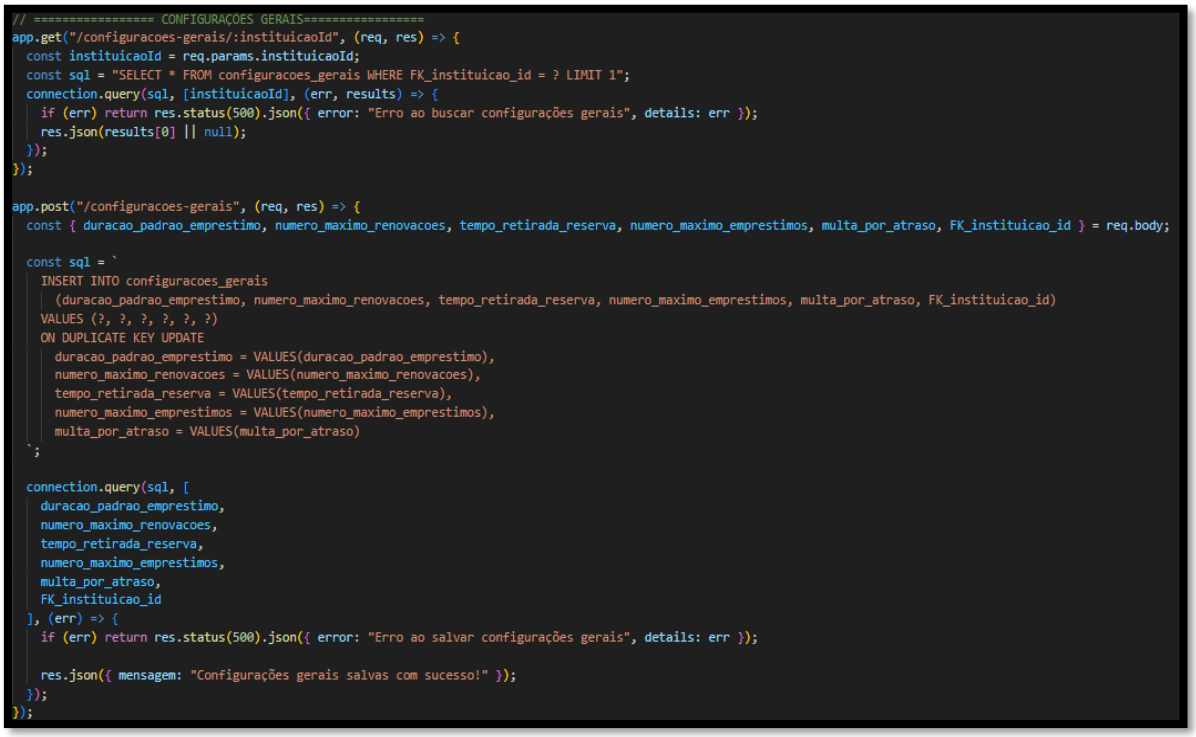
Ao carregar a página, o evento *DOMContentLoaded* é acionado, vinculando a função ao botão de salvar e executando a função carregarConfiguracoesGerais. Essa

função realiza uma requisição HTTP *GET* à API, buscando as configurações já cadastradas no banco de dados com base no ID da instituição do funcionário logado. Os valores recebidos são inseridos automaticamente nos campos do formulário, garantindo que o administrador visualize e, se necessário, atualize os dados.

Já a função salvarConfiguracoesGerais é responsável por enviar os ajustes feitos. Os valores digitados são organizados em um objeto dados, convertido em JSON e transmitido ao servidor por meio de uma requisição *POST*. Caso o salvamento seja bem-sucedido, o sistema exibe uma mensagem de confirmação e recarrega as configurações atualizadas. Em caso de falha, é mostrado um alerta informando o erro ao usuário.

Assim, o código do *front-end* assegura a interação dinâmica entre interface e servidor, mantendo os campos sincronizados e oferecendo *feedback* imediato sobre o processo de configuração.

Figura 104 - Conexão com banco de dados configuração de prazos e regras



```
// ================ CONFIGURAÇÕES GERAIS================
app.get("/configuracoes-gerais/:instituicaoId", (req, res) => {
  const instituicaoId = req.params.instituicaoId;
  const sql = "SELECT * FROM configuracoes_gerais WHERE FK_instituicao_id = ? LIMIT 1";
  connection.query(sql, [instituicaoId], (err, results) => {
    if (err) return res.status(500).json({ error: "Erro ao buscar configurações gerais", details: err });
    res.json(results[0] || null);
  });
});

app.post("/configuracoes-gerais", (req, res) => {
  const { duracao_padrao_emprestimo, numero_maximo_renovacoes, tempo_retirada_reserva, numero_maximo_emprestimos, multa_por_atraso, FK_instituicao_id } = req.body;

  const sql = `
    INSERT INTO configuracoes_gerais
      (duracao_padrao_emprestimo, numero_maximo_renovacoes, tempo_retirada_reserva, numero_maximo_emprestimos, multa_por_atraso, FK_instituicao_id)
    VALUES (?, ?, ?, ?, ?, ?)
    ON DUPLICATE KEY UPDATE
      duracao_padrao_emprestimo = VALUES(duracao_padrao_emprestimo),
      numero_maximo_renovacoes = VALUES(numero_maximo_renovacoes),
      tempo_retirada_reserva = VALUES(tempo_retirada_reserva),
      numero_maximo_emprestimos = VALUES(numero_maximo_emprestimos),
      multa_por_atraso = VALUES(multa_por_atraso)
  `;

  connection.query(sql, [
    duracao_padrao_emprestimo,
    numero_maximo_renovacoes,
    tempo_retirada_reserva,
    numero_maximo_emprestimos,
    multa_por_atraso,
    FK_instituicao_id
  ], (err) => {
    if (err) return res.status(500).json({ error: "Erro ao salvar configurações gerais", details: err });

    res.json({ mensagem: "Configurações gerais salvas com sucesso!" });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

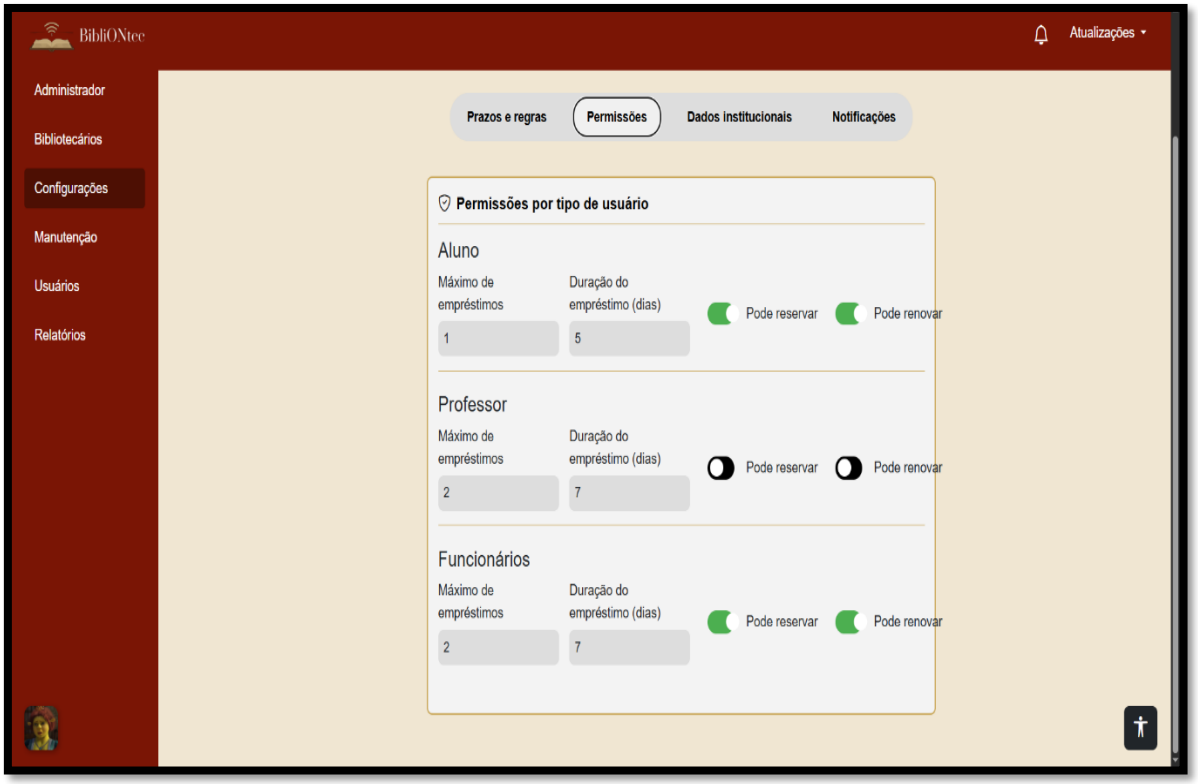
O código exibido na Figura 104 implementa as rotas de *back-end* responsáveis por buscar e salvar as configurações gerais no banco de dados MySQL, utilizando o *framework Express*.

A rota *GET* /configuracoes-gerais/:instituicaoId recebe como parâmetro o ID da instituição e executa uma consulta SQL para recuperar os registros da tabela configuracoes_gerais. Caso os dados existam, são enviados em formato JSON para o *front-end*; se não houver registros, o sistema retorna *null*. Em caso de erro de consulta, é retornado o *status* 500 com detalhes da falha.

Já a rota *POST* /configuracoes-gerais é responsável por inserir ou atualizar os dados. A instrução SQL utiliza a cláusula *ON DUPLICATE KEY UPDATE*, o que permite tanto criar uma nova configuração quanto atualizar os valores já existentes, sem duplicação. Os parâmetros recebidos incluem duração de empréstimo, número máximo de renovações, tempo para retirada de reservas, limite de empréstimos por usuário, multa por atraso e o ID da instituição.

Caso a operação seja concluída com sucesso, o servidor retorna uma mensagem confirmando o salvamento. Se ocorrer erro durante a execução, é retornada uma resposta com *status* 500 e detalhes do problema.

Figura 105 - Tela configuração de permissões



FONTE: AUTORIA PRÓPRIA (2025)

A tela de Permissões por Tipo de Usuário, mostrada na figura 105, tem como finalidade permitir que o administrador configure de forma individualizada as regras de empréstimo de acordo com o perfil do usuário da biblioteca. A interface é dividida em blocos, cada um representando um grupo específico: Alunos, Professores e Funcionários. Dentro de cada bloco, é possível definir parâmetros importantes como o número máximo de empréstimos permitidos, a duração padrão desses empréstimos em dias, além de permissões extras controladas por botões de alternância, que indicam se o usuário pode ou não realizar reservas e renovações.

Essa personalização é fundamental porque reconhece as diferentes necessidades dos perfis que utilizam a biblioteca. Alunos, por exemplo, geralmente têm prazos menores e menos empréstimos simultâneos, enquanto professores podem necessitar de mais tempo e maior quantidade de exemplares devido às atividades de ensino e pesquisa. Já os funcionários podem ter regras intermediárias, equilibrando flexibilidade e controle.

No canto superior direito da tela, há o botão Salvar alterações, responsável por consolidar todas as mudanças feitas e aplicá-las no sistema. A interface foi pensada para ser prática e objetiva, permitindo ajustes rápidos sem necessidade de navegação adicional. Essa abordagem garante que a gestão da biblioteca se torne mais dinâmica, flexível e adaptada à realidade dos diferentes públicos atendidos.

Além disso, essa estrutura possibilita maior transparência na aplicação das regras. Do ponto de vista administrativo, centralizar essas configurações em uma única tela otimiza o trabalho dos bibliotecários e reduz a chance de inconsistências nos registros de empréstimos.

Outro ponto relevante é que essas permissões podem ser ajustadas de acordo com mudanças institucionais ou demandas específicas, como períodos de férias, semanas de provas ou projetos de pesquisa em andamento. Assim, o sistema torna-se não apenas uma ferramenta de controle, mas também de apoio às atividades acadêmicas e administrativas.

Figura 106 - Código configuração de permissões



```
document.addEventListener("DOMContentLoaded", () => {
  const salvarBtn = document.querySelector(".btn-save");
  const funcionarioLogado = JSON.parse(localStorage.getItem("funcionario"));
  if (!funcionarioLogado) return console.error("Nenhum funcionário logado.");
  const instituicaoId = funcionarioLogado.FK_instituicao_id;
  // ---------- CARREGAR CONFIGURAÇÕES ----------
  async function carregarTipoUsuario() {
    try {
      const res = await fetch(`http://localhost:3000/configuracoes-tipo-usuario/${instituicaoId}`);
      const configs = await res.json();

      const blocos = document.querySelectorAll(".user-role"); // todos os blocos: Aluno, Professor, Funcionários

      configs.forEach(cfg => {
        const index = cfg.FK_tipo_usuario_id - 1; // 1=aluno, 2=professor, 3=funcionários
        const bloco = blocos[index];
        if (!bloco) return;

        const inputs = bloco.querySelectorAll("input[type=number]");
        const checks = bloco.querySelectorAll("input[type=checkbox]");

        if (inputs.length >= 2) {
          inputs[0].value = cfg.maximo_emprestimos || 0;
          inputs[1].value = cfg.duracao_emprestimo || 0;
        }
        if (checks.length >= 2) {
          checks[0].checked = cfg.pode_reservar === 1;
          checks[1].checked = cfg.pode_renovar === 1;
        }

        bloco.dataset.id = cfg.id;
      });
    } catch (err) {
      console.error("Erro ao carregar tipo usuário:", err);
    }
  }
  console.log(localStorage.getItem("funcionario"));
  // ---------- SALVAR CONFIGURAÇÕES ----------
  async function salvarTipoUsuario() {
    const blocos = document.querySelectorAll(".user-role"); // todos os blocos
    const tipos = [1, 2, 3]; // mapeia: 1=Aluno, 2=Professor, 3=Funcionários

    for (let idx = 0; idx < blocos.length; idx++) {
      const el = blocos[idx];
      const inputs = el.querySelectorAll("input[type=number]");
      const checks = el.querySelectorAll("input[type=checkbox]");

      const dados = {
        FK_tipo_usuario_id: tipos[idx],
        maximo_emprestimos: parseInt(inputs[0]?.value) || 0,
        duracao_emprestimo: parseInt(inputs[1]?.value) || 0,
        pode_reservar: checks[0]?.checked ? 1 : 0,
        pode_renovar: checks[1]?.checked ? 1 : 0,
        FK_instituicao_id: instituicaoId
      };

      const id = el.dataset.id;

      try {
        const resposta = await fetch(
          id
            ? `http://localhost:3000/configuracoes-tipo-usuario/${id}`
            : "http://localhost:3000/configuracoes-tipo-usuario",
          {
            method: id ? "PUT" : "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify(dados)
          }
        );

        const resultado = await resposta.json();
        console.log("✔", resultado.mensagem);
      } catch (erro) {
        console.error("Erro ao salvar tipo usuário:", erro);
      }
    }

    alert("✅ Configurações de tipos de usuário salvas!");
    carregarTipoUsuario();
  }

  // ---------- INICIALIZA ----------
  carregarTipoUsuario();
  if (salvarBtn) salvarBtn.addEventListener("click", salvarTipoUsuario);
});
```

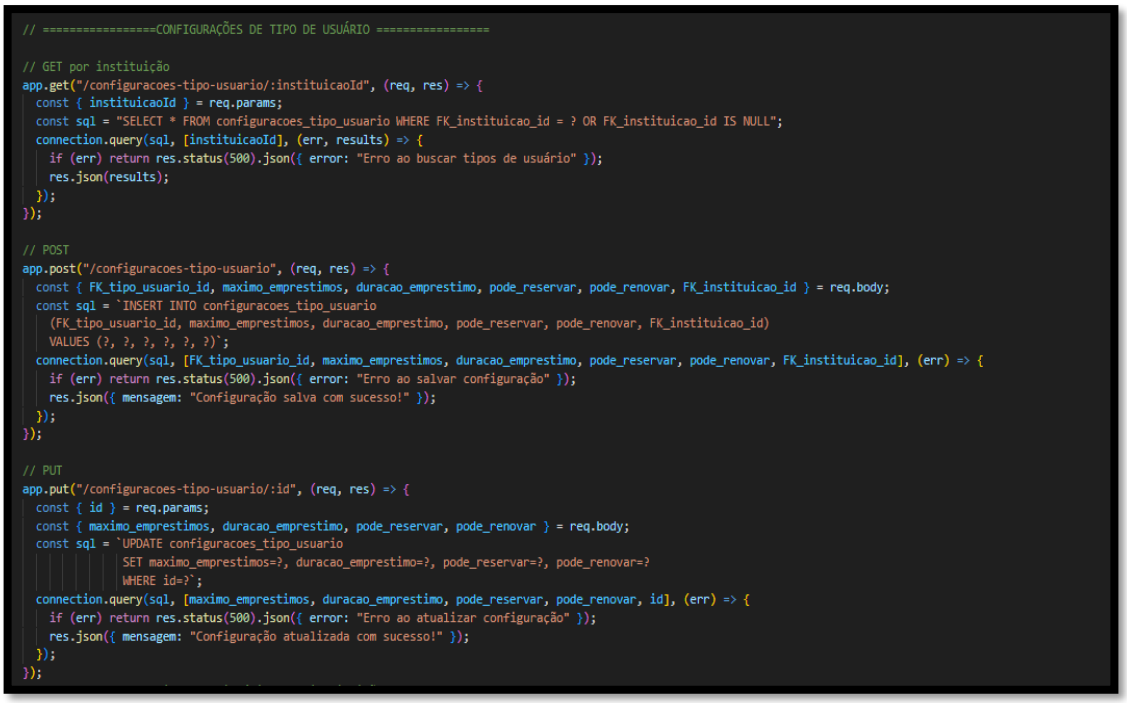
FONTE: AUTORIA PRÓPRIA (2025)

O código do *front-end* que implementa a tela de configuração de permissões, visto na figura 106, é responsável por carregar e salvar as configurações de cada tipo de usuário. No carregamento inicial, a função carregarTipoUsuario() faz uma requisição ao servidor para buscar as permissões já cadastradas. Os dados retornados são distribuídos pelos blocos da interface, preenchendo os campos

numéricos e ativando ou desativando os botões de alternância conforme o perfil. Cada bloco recebe também o identificador do registro no banco de dados, armazenado como atributo para facilitar futuras atualizações.

No momento de salvar, a função salvarTipoUsuario() percorre cada bloco e monta um objeto com os dados preenchidos pelo administrador. Caso já exista um registro vinculado, é feita uma requisição *PUT* para atualizar; se não existir, utiliza-se um *POST* para criar um novo. Esse processo é repetido para todos os tipos de usuário, garantindo que cada perfil receba suas próprias configurações de forma independente. Ao final, uma mensagem confirma a conclusão do processo, e a interface é recarregada para refletir os novos dados.

Figura 107 - Conexão com banco de dados configuração de permissões



```
// ================CONFIGURAÇÕES DE TIPO DE USUÁRIO ================

// GET por instituição
app.get("/configuracoes-tipo-usuario/:instituicaoId", (req, res) => {
  const { instituicaoId } = req.params;
  const sql = "SELECT * FROM configuracoes_tipo_usuario WHERE FK_instituicao_id = ? OR FK_instituicao_id IS NULL";
  connection.query(sql, [instituicaoId], (err, results) => {
    if (err) return res.status(500).json({ error: "Erro ao buscar tipos de usuário" });
    res.json(results);
  });
});

// POST
app.post("/configuracoes-tipo-usuario", (req, res) => {
  const { FK_tipo_usuario_id, maximo_emprestimos, duracao_emprestimo, pode_reservar, pode_renovar, FK_instituicao_id } = req.body;
  const sql = `INSERT INTO configuracoes_tipo_usuario
    (FK_tipo_usuario_id, maximo_emprestimos, duracao_emprestimo, pode_reservar, pode_renovar, FK_instituicao_id)
    VALUES (?, ?, ?, ?, ?, ?)`;
  connection.query(sql, [FK_tipo_usuario_id, maximo_emprestimos, duracao_emprestimo, pode_reservar, pode_renovar, FK_instituicao_id], (err) => {
    if (err) return res.status(500).json({ error: "Erro ao salvar configuração" });
    res.json({ mensagem: "Configuração salva com sucesso!" });
  });
});

// PUT
app.put("/configuracoes-tipo-usuario/:id", (req, res) => {
  const { id } = req.params;
  const { maximo_emprestimos, duracao_emprestimo, pode_reservar, pode_renovar } = req.body;
  const sql = `UPDATE configuracoes_tipo_usuario
               SET maximo_emprestimos=?, duracao_emprestimo=?, pode_reservar=?, pode_renovar=?
               WHERE id=?`;
  connection.query(sql, [maximo_emprestimos, duracao_emprestimo, pode_reservar, pode_renovar, id], (err) => {
    if (err) return res.status(500).json({ error: "Erro ao atualizar configuração" });
    res.json({ mensagem: "Configuração atualizada com sucesso!" });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

No *back-end*, visto na figura 107, a lógica foi implementada por meio de três rotas principais. A primeira, um *GET*, busca todas as configurações vinculadas a uma instituição específica, podendo ainda retornar registros globais (sem instituição definida) que funcionam como padrão. A segunda, via *POST*, permite inserir novas configurações, gravando no banco o máximo de empréstimos, duração, e as permissões de reserva e renovação para determinado tipo de usuário. Por fim, a rota

PUT possibilita atualizar registros já existentes, recebendo o identificador da configuração e os novos valores que devem ser aplicados.

Essa separação garante que a tela de permissões consiga manipular tanto novos cadastros quanto alterações de regras já existentes. Diferente da tela de regras gerais, que trabalha com um único conjunto de informações, a aba de permissões exige que múltiplos registros sejam tratados em paralelo, um para cada categoria de usuário, oferecendo maior flexibilidade e adequação às necessidades da biblioteca.

Figura 108 - Tela configurações da instituição



FONTE: AUTORIA PRÓPRIA (2025)

A aba Dados Institucionais, mostrada na figura 108, tem como objetivo armazenar e disponibilizar as principais informações sobre a instituição à qual o sistema está vinculado. Essa seção funciona como um cadastro central de dados institucionais, reunindo campos como: nome da instituição, *e-mail*, horário de funcionamento, telefone, *site*, endereço e tipo de instituição.

O formulário é simples e direto, permitindo ao administrador inserir ou atualizar esses dados de forma prática. Um destaque importante é o campo Tipo de Instituição, exibido como um seletor, garantindo que os valores disponíveis sejam previamente cadastrados no sistema.

O botão Salvar alterações, localizado no canto superior direito, é responsável por enviar os dados preenchidos ou modificados para o servidor, que os grava no banco de dados. Caso já exista um registro, ele é atualizado; se não, um novo cadastro é criado.

Figura 109 - Código configurações da instituição



```
// Carregar os tipos de instituição no select
async function carregarTiposInstituicao() {
    try {
      const resposta = await fetch("http://localhost:3000/tipos-instituicao");
      if (!resposta.ok) throw new Error("Falha ao buscar tipos de instituição");

      const tipos = await resposta.json();

      const select = document.getElementById("institutionType");
      select.innerHTML = '<option value="">Selecione o tipo</option>';

      tipos.forEach(t => {
        const option = document.createElement("option");
        option.value = t.id;
        option.textContent = t.tipo;
        select.appendChild(option);
      });
    } catch (error) {
      console.error("Erro ao carregar tipos de instituição:", error);
      alert("Não foi possível carregar os tipos de instituição.");
    }
  }

  // Carregar instituição existente
  async function carregarInstituicao() {
    try {
      const resposta = await fetch("http://localhost:3000/instituicao");
      const instituicao = await resposta.json();

      if (instituicao) {
        document.getElementById("institutionName").value = instituicao.nome || "";
        document.getElementById("email").value = instituicao.email || "";
        document.getElementById("workingHours").value = instituicao.horario_funcionamento || "";
        document.getElementById("phone").value = instituicao.telefone || "";
        document.getElementById("website").value = instituicao.website || "";
        document.getElementById("address").value = instituicao.endereco || "";
        document.getElementById("institutionType").value = instituicao.FK_tipo_instituicao_id || "";

        // 🔹 Guarda o ID diretamente no SELECT (já que não temos form)
        document.getElementById("institutionType").dataset.id = instituicao.id;
      }
    } catch (error) {
      console.error("Erro ao carregar instituição:", error);
    }
  }

  // Quando a página carregar
  document.addEventListener("DOMContentLoaded", () => {
    carregarTiposInstituicao().then(() => carregarInstituicao());

    const btnSalvar = document.querySelector(".btn-save"); // seu botão já existe no HTML
    btnSalvar.addEventListener("click", async () => {
      const dados = {
        nome: document.getElementById("institutionName").value,
        email: document.getElementById("email").value,
        horario_funcionamento: document.getElementById("workingHours").value,
        telefone: document.getElementById("phone").value,
        website: document.getElementById("website").value,
        endereco: document.getElementById("address").value,
        FK_tipo_instituicao_id: document.getElementById("institutionType").value
      };
```

FONTE: AUTORIA PRÓPRIA (2025)

O código do *front-end* que controla esta tela, visto na figura 109, é responsável por carregar e salvar as informações das instituições. No carregamento inicial, a função carregarTiposInstituicao() faz uma requisição ao servidor para buscar os tipos de instituição cadastrados. Os dados retornados são utilizados para preencher o campo <*select*>, garantindo que o administrador escolha apenas opções válidas e padronizadas.

Em seguida, a função carregarInstituicao() consulta a API para verificar se já existe um registro da instituição. Caso haja, os campos do formulário são preenchidos automaticamente com os dados armazenados no banco, e o ID do registro é salvo como atributo no seletor de tipo, permitindo diferenciar entre inserção e atualização.

No momento de salvar, a função responsável reúne todas as informações preenchidas pelo administrador e monta um objeto dados. Se já existir um ID, é feita uma requisição *PUT* para atualizar o registro; caso contrário, utiliza-se um *POST* para criar um novo. Após a operação, uma mensagem confirma o sucesso do processo, e os dados são recarregados para manter a tela atualizada.

Figura 110 - Conexão com banco de dados configurações da instituição



```
// ---------------- Lista tipos de instituição ----------------

app.get("/tipos-instituicao", (req, res) => {
  const sql = "SELECT * FROM tipo_instituicao";
  connection.query(sql, (err, results) => {
    if (err) {
      console.error(err);
      return res.status(500).json({ mensagem: "Erro ao carregar tipos de instituição" });
    }
    res.json(results);
  });
});

// Buscar dados da instituição
app.get("/instituicao", (req, res) => {
  const sql = "SELECT * FROM instituicao LIMIT 1";
  connection.query(sql, (err, results) => {
    if (err) {
      console.error("Erro ao buscar instituição:", err);
      return res.status(500).json({ mensagem: "Erro ao buscar instituição" });
    }
    if (results.length === 0) {
      return res.json(null);
    }
    res.json(results[0]);
  });
});

//  ---------------- Cadastrar instituição ----------------

app.post("/instituicao", (req, res) => {
  const { nome, email, horario_funcionamento, telefone, website, endereco, FK_tipo_instituicao_id } = req.body;

  if (!nome || !FK_tipo_instituicao_id) {
    return res.status(400).json({ mensagem: "Nome e tipo da instituição são obrigatórios." });
  }

  const sql = `
      INSERT INTO instituicao
      (nome, email, horario_funcionamento, telefone, website, endereco, FK_tipo_instituicao_id)
      VALUES (?, ?, ?, ?, ?, ?, ?)
    `;

  connection.query(sql, [nome, email, horario_funcionamento, telefone, website, endereco, FK_tipo_instituicao_id], (err, result) => {
    if (err) {
      console.error("Erro ao cadastrar instituição:", err);
      return res.status(500).json({ mensagem: "Erro ao cadastrar instituição" });
    }
    res.json({ mensagem: "Instituição cadastrada com sucesso!", id: result.insertId });
  });
});

// Atualizar instituição
app.put("/instituicao/:id", (req, res) => {
  const { id } = req.params;
  const { nome, email, horario_funcionamento, telefone, website, endereco, FK_tipo_instituicao_id } = req.body;

  const sql = `
    UPDATE instituicao
    SET nome = ?, email = ?, horario_funcionamento = ?, telefone = ?, website = ?, endereco = ?, FK_tipo_instituicao_id = ?
    WHERE id = ?
  `;

  connection.query(sql, [nome, email, horario_funcionamento, telefone, website, endereco, FK_tipo_instituicao_id, id], (err) => {
    if (err) {
      console.error("Erro ao atualizar instituição:", err);
      return res.status(500).json({ mensagem: "Erro ao atualizar instituição" });
    }
    res.json({ mensagem: "Instituição atualizada com sucesso!" });
  });
```

FONTE: AUTORIA PRÓPRIA (2025)

No *back-end*, visto na figura 110, a lógica de persistência e consulta dos dados institucionais foi implementada por meio de quatro rotas principais. A primeira, um *GET* em /tipos-instituicao, retorna todos os registros da tabela tipo_instituicao, utilizados para preencher o seletor de tipos no *front-end*. A segunda, um *GET* em /instituicao, recupera o primeiro registro da tabela instituicao e retorna *null* caso não exista nenhum, permitindo pré-preencher o formulário ao carregar a tela.

A terceira rota, via *POST* em /instituicao, recebe os dados enviados pelo *front-end*, valida campos obrigatórios como nome e tipo, e insere um novo registro no banco de dados. Em caso de sucesso, retorna uma mensagem de confirmação junto com o ID do registro criado. Por fim, a rota *PUT* em /instituicao/:id possibilita atualizar um registro já existente com os novos valores fornecidos, utilizando o ID informado na rota; erros de banco resultam em *status* 500 acompanhado de uma mensagem de erro.

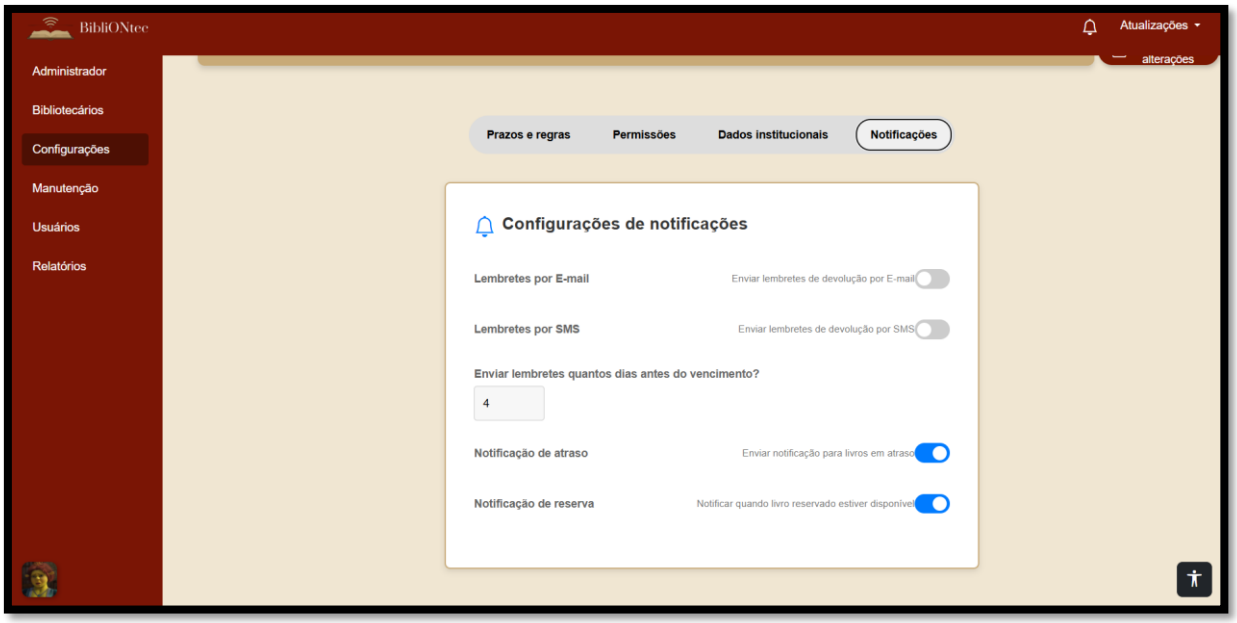
Essa organização garante que a aba de Dados Institucionais possa manipular tanto novos cadastros quanto atualizações de registros existentes, mantendo os dados sempre consistentes, validados e corretamente persistidos no banco, de forma transparente para o administrador.

Outro aspecto importante é que as consultas e atualizações são protegidas por validações básicas, reduzindo a chance de falhas ou inserções incompletas. Isso torna o fluxo mais confiável e evita que informações críticas, como *e-mail* e telefone da instituição, sejam registradas de forma incorreta.

Além disso, a lógica implementada permite escalabilidade: caso seja necessário incluir novas propriedades da instituição no futuro (como logotipo ou redes sociais), o modelo pode ser facilmente expandido sem comprometer a estrutura já existente.

Do ponto de vista prático, o uso de respostas claras em JSON favorece a integração direta com o *front-end*, que consegue exibir mensagens amigáveis e reativas ao usuário final. Esse formato também torna mais simples a manutenção do código e o rastreamento de problemas.

Figura 111 - Tela configurações de notificação



FONTE: AUTORIA PRÓPRIA (2025)

A Figura 111 apresenta a tela de configurações de notificações, cuja finalidade é oferecer ao administrador um espaço para definir como os usuários da biblioteca serão lembrados sobre prazos e movimentações de empréstimos.

Nessa interface, estão disponíveis opções para ativar ou desativar lembretes enviados por *e-mail* e SMS, configurando o canal de comunicação mais adequado para a instituição. Além disso, há um campo que permite determinar com quantos dias de antecedência os alertas devem ser disparados antes do vencimento da devolução.

Outro ponto importante são as notificações automáticas de atraso e de reserva, que podem ser ativadas individualmente. Dessa forma, o administrador garante que os usuários sejam avisados quando houver pendências ou quando um livro reservado estiver disponível para retirada.

O *layout* simples, baseado em interruptores e caixas numéricas, contribui para uma navegação prática e intuitiva, facilitando o gerenciamento das configurações sem necessidade de conhecimento técnico avançado.

Figura 112 - Código configurações de notificação



```
document.addEventListener("DOMContentLoaded", () => {
  const salvarBtn = document.querySelector(".btn-save");
  const funcionarioLogado = JSON.parse(localStorage.getItem("funcionario"));
  if (!funcionarioLogado) return console.error("Nenhum funcionário logado.");

  const instituicaoId = funcionarioLogado.FK_instituicao_id;

  async function carregarNotificacoes() {
    try {
      const res = await fetch(`http://localhost:3000/configuracoes-notificacao/${instituicaoId}`);
      const config = await res.json();

      const email = document.getElementById("emailNotifications");
      const sms = document.getElementById("smsNotifications");
      const dias = document.getElementById("reminderDays"); // corrigido
      const atraso = document.getElementById("overdueNotification"); // corrigido
      const reserva = document.getElementById("reservationNotification"); // corrigido

      if (!email || !sms || !dias || !atraso || !reserva) {
        console.warn("⚠ Campos de notificação não encontrados no HTML.");
        return;
      }

      if (config) {
        email.checked = config.lembrete_vencimento === 1;
        sms.checked = config.sms_notificacao === 1;
        dias.value = config.dias_antes_vencimento || 0;
        atraso.checked = config.notificacao_atraso === 1;
        reserva.checked = config.notificacao_reserva === 1;

        email.dataset.id = config.id;
      }
    } catch (err) {
      console.error("Erro ao carregar notificações:", err);
    }
  }

  async function salvarNotificacoes() {
    const email = document.getElementById("emailNotifications");
    const sms = document.getElementById("smsNotifications");
    const dias = document.getElementById("reminderDays");
    const atraso = document.getElementById("overdueNotification");
    const reserva = document.getElementById("reservationNotification");

    if (!email || !sms || !dias || !atraso || !reserva) {
      return alert("⚠ Não encontrei os campos de notificação no HTML.");
    }

    const dados = {
      lembrete_vencimento: email.checked ? 1 : 0,
      sms_notificacao: sms.checked ? 1 : 0,
      dias_antes_vencimento: parseInt(dias.value) || 0,
      notificacao_atraso: atraso.checked ? 1 : 0,
      notificacao_reserva: reserva.checked ? 1 : 0,
      notificacao_livro_disponivel: 1,
      FK_instituicao_id: instituicaoId
    };

    const id = email.dataset.id;

    try {
      const resposta = await fetch(
        id
          ? `http://localhost:3000/configuracoes-notificacao/${id}`
          : "http://localhost:3000/configuracoes-notificacao",
        {
          method: id ? "PUT" : "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify(dados)
        }
      );

      const resultado = await resposta.json();
      alert(resultado.mensagem || "✅ Notificações salvas!");
      carregarNotificacoes();
    } catch (erro) {
      console.error("Erro ao salvar notificações:", erro);
      alert("❌ Não foi possível salvar as notificações.");
    }
  }

  carregarNotificacoes();
  if (salvarBtn) salvarBtn.addEventListener("click", salvarNotificacoes);
```

FONTE: AUTORIA PRÓPRIA (2025)

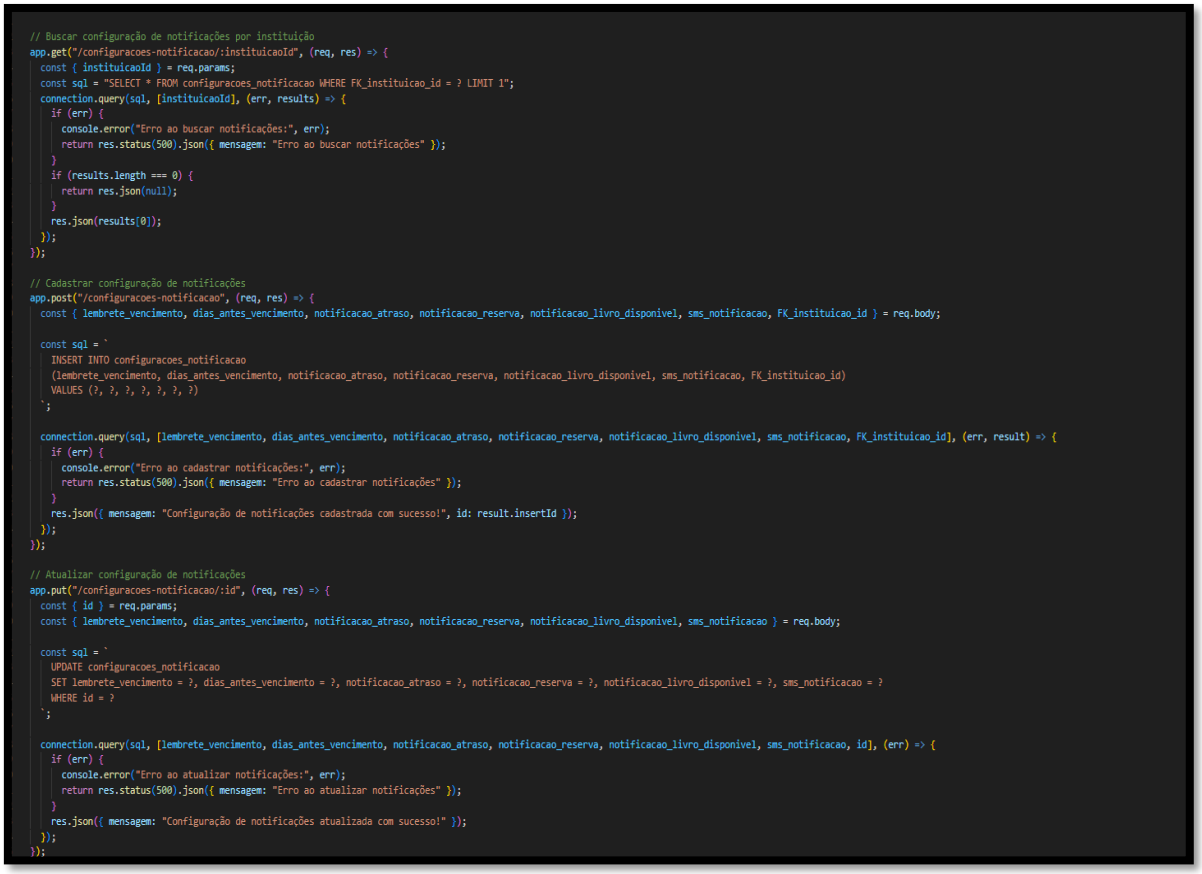
O código apresentado na Figura 112 corresponde à lógica do *front-end* responsável por carregar e salvar as configurações de notificações no sistema.

Ao carregar a página, o evento *DOMContentLoaded* é acionado, inicializando a função carregarNotificacoes. Essa função realiza uma requisição HTTP *GET* ao servidor, recuperando os dados já registrados para a instituição vinculada ao funcionário logado. Caso haja valores cadastrados, eles são preenchidos automaticamente nos campos da tela, garantindo que a interface reflita fielmente o estado atual das configurações.

Já a função salvarNotificacoes organiza os ajustes feitos pelo administrador em um objeto JSON e envia as informações ao *back-end*. Quando ainda não existe configuração registrada, é utilizada uma requisição *POST*; caso já haja registros, o envio é feito por meio de uma requisição *PUT*.

Após o salvamento, o sistema exibe uma mensagem de confirmação e recarrega as informações atualizadas, assegurando que o administrador visualize imediatamente o resultado das alterações. Em caso de falha, mensagens de erro são exibidas, mantendo a transparência e a confiabilidade do processo.

Figura 113 - Conexão com banco de dados configurações de notificação



```
// Buscar configuração de notificações por instituição
app.get("/configuracoes-notificacao/:instituicaoId", (req, res) => {
  const { instituicaoId } = req.params;
  const sql = "SELECT * FROM configuracoes_notificacao WHERE FK_instituicao_id = ? LIMIT 1";
  connection.query(sql, [instituicaoId], (err, results) => {
    if (err) {
      console.error("Erro ao buscar notificações:", err);
      return res.status(500).json({ mensagem: "Erro ao buscar notificações" });
    }
    if (results.length === 0) {
      return res.json(null);
    }
    res.json(results[0]);
  });
});

// Cadastrar configuração de notificações
app.post("/configuracoes-notificacao", (req, res) => {
  const { lembrete_vencimento, dias_antes_vencimento, notificacao_atraso, notificacao_reserva, notificacao_livro_disponivel, sms_notificacao, FK_instituicao_id } = req.body;

  const sql = `
    INSERT INTO configuracoes_notificacao
    (lembrete_vencimento, dias_antes_vencimento, notificacao_atraso, notificacao_reserva, notificacao_livro_disponivel, sms_notificacao, FK_instituicao_id)
    VALUES (?, ?, ?, ?, ?, ?, ?)
  `;

  connection.query(sql, [lembrete_vencimento, dias_antes_vencimento, notificacao_atraso, notificacao_reserva, notificacao_livro_disponivel, sms_notificacao, FK_instituicao_id], (err, result) => {
    if (err) {
      console.error("Erro ao cadastrar notificações:", err);
      return res.status(500).json({ mensagem: "Erro ao cadastrar notificações" });
    }
    res.json({ mensagem: "Configuração de notificações cadastrada com sucesso!", id: result.insertId });
  });
});

// Atualizar configuração de notificações
app.put("/configuracoes-notificacao/:id", (req, res) => {
  const { id } = req.params;
  const { lembrete_vencimento, dias_antes_vencimento, notificacao_atraso, notificacao_reserva, notificacao_livro_disponivel, sms_notificacao } = req.body;

  const sql = `
    UPDATE configuracoes_notificacao
    SET lembrete_vencimento = ?, dias_antes_vencimento = ?, notificacao_atraso = ?, notificacao_reserva = ?, notificacao_livro_disponivel = ?, sms_notificacao = ?
    WHERE id = ?
  `;

  connection.query(sql, [lembrete_vencimento, dias_antes_vencimento, notificacao_atraso, notificacao_reserva, notificacao_livro_disponivel, sms_notificacao, id], (err) => {
    if (err) {
      console.error("Erro ao atualizar notificações:", err);
      return res.status(500).json({ mensagem: "Erro ao atualizar notificações" });
    }
    res.json({ mensagem: "Configuração de notificações atualizada com sucesso!" });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

No *back-end*, apresentado na figura 113, a implementação é feita por meio de três rotas que asseguram a persistência das configurações de notificação no banco de dados.

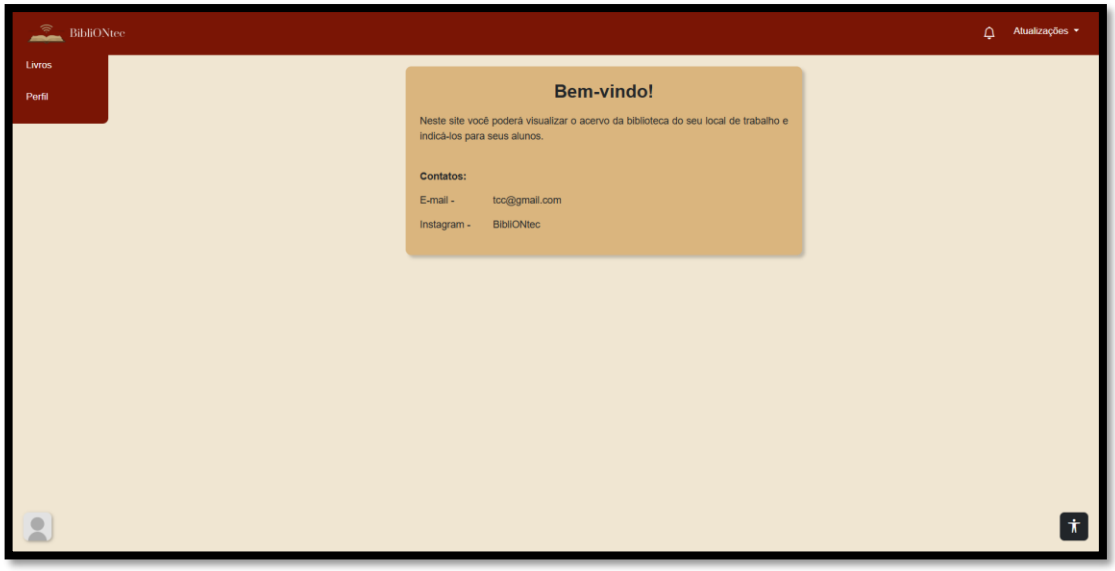
A rota *GET* /configuracoes-notificacao/:instituicaoId é responsável por buscar as informações associadas a uma instituição específica. Se não houver registros, retorna *null*, permitindo que o *front-end* saiba que ainda não existem dados configurados.

A rota *POST* /configuracoes-notificacao realiza a inserção de uma nova configuração. Nela, são cadastrados parâmetros como lembrete de vencimento, envio por SMS, número de dias de antecedência e ativação de notificações de atraso ou reserva. Já a rota *PUT* /configuracoes-notificacao/:id permite atualizar uma configuração existente, sobrescrevendo os valores antigos com as novas preferências definidas pelo administrador.

Cada uma dessas rotas conta com tratamento de erros e retorna mensagens claras em formato JSON, facilitando a comunicação entre *front-end* e servidor. Dessa maneira, o código do *back-end* garante que as regras de notificação sejam corretamente aplicadas e armazenadas, mantendo a integridade e a eficiência do sistema.

3.4.15 – Tela Inicial do professor

Figura 114 - Tela inicial do professor



FONTE: AUTORIA PRÓPRIA (2025)

A figura 114 apresenta a tela inicial do professor no sistema BibliONtec. Essa interface tem como objetivo oferecer uma visão de boas-vindas e orientar o usuário sobre as funcionalidades disponíveis.

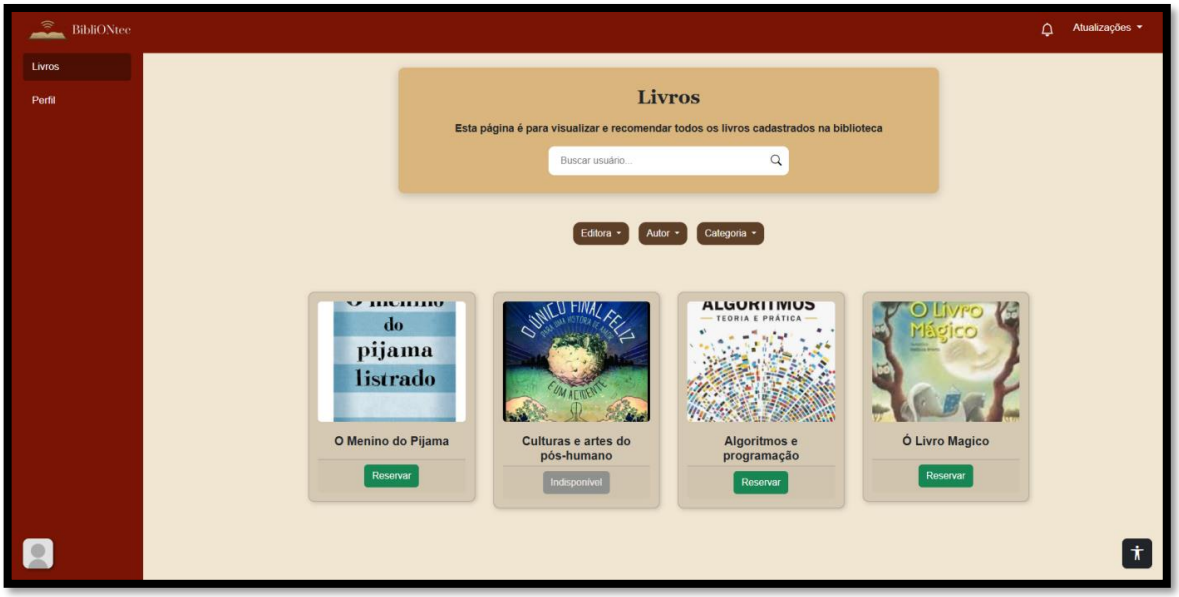
Na parte central, em destaque, há um quadro em tom bege com a mensagem “Bem-vindo!”, seguida de um breve texto explicativo que informa que o professor pode utilizar o *site* para visualizar o acervo da biblioteca de sua instituição e indicar livros para seus alunos.

Ainda nesse quadro, são apresentados os contatos institucionais, incluindo um endereço de *e-mail* e o perfil oficial do sistema no Instagram, permitindo uma comunicação direta entre os usuários e a administração da plataforma.

No canto esquerdo da tela, encontra-se o menu lateral de navegação, em vermelho, contendo as opções “Livros” e “Perfil”, que direcionam o professor às seções principais de sua área no sistema.

3.4.16 – Área de livros do professor

Figura 115 - Tela do professor área dos livros



FONTE: AUTORIA PRÓPRIA (2025)

A figura 115 apresenta a área do professor no módulo Livros do sistema BibliONtec. Nessa interface, o docente pode visualizar todo o acervo disponível na biblioteca, bem como indicar obras para seus alunos ou reserva-los para si. No topo da página há um campo de busca que permite filtrar os livros de acordo com o termo

digitado, facilitando a localização de títulos específicos. Logo abaixo, existem botões que possibilitam organizar a visualização do acervo por editora, autor ou categoria, oferecendo maior flexibilidade na consulta.

As obras são exibidas em formato de cartões, cada um contendo a capa do livro, seu título e o botão de ação correspondente. No exemplo exibido, alguns livros aparecem com a opção “Reservar”, sinalizada em verde, enquanto outros apresentam o *status* “Indisponível”, indicando que já estão em uso. Essa diferenciação visual garante que o professor identifique rapidamente quais exemplares podem ser reservados para si, ou indicados para seus alunos. O código *front-end* e *back-end* seguem o mesmo padrão apresentado nas figuras 83, 84 e 86, com a diferença de não haver alteração e exclusão dos livros.

3.4.17 – Área do professor

Figura 116 - Tela área do professor



FONTE: AUTORIA PRÓPRIA (2025)

A figura 116 mostra a tela correspondente à área do professor no sistema BibliONtec e foi projetada para oferecer ao docente um espaço personalizado de gerenciamento de sua conta e de acompanhamento das atividades relacionadas à biblioteca. Logo no topo, encontram-se os campos de dados pessoais já preenchidos com as informações do professor, como nome, *e-mail*, telefone e senha, permitindo que ele visualize e edite seus dados por meio do botão “Editar Perfil”. Essa

funcionalidade garante que as informações cadastradas no sistema possam ser sempre mantidas atualizadas, reforçando a confiabilidade do banco de dados.

Ao centro da tela, há um botão em destaque, “Notificações atualizadas!”, que funciona como um recurso de alerta rápido para informar o professor sobre as movimentações mais recentes em sua conta, como prazos de entrega e livros disponíveis.

Na parte inferior, existe um quadro de notificações com o título “Livros que vão ser entregues hoje”. Nele, são listadas as obras que devem ser devolvidas na data atual, apresentadas com detalhes que incluem o nome do livro, a data da entrega e os temas discutidos. Esse recurso facilita o acompanhamento da rotina de empréstimos, permitindo ao professor se organizar melhor quanto às devoluções.

O *design* segue o padrão visual do sistema, com tons de bege e marrom, mantendo a identidade visual do BibliONtec e a organização dos elementos de forma clara e objetiva. Assim, essa tela oferece ao professor praticidade na atualização de seus dados e no controle de prazos. O código *front-end* segue o mesmo padrão do usado nas outras áreas, nas figuras 56 e 57.

Figura 117 - Conexão com o banco de dados da tela do professor

```
// ================ Rota: atualizar usuário ================
app.put('/api/usuarios/:id', autenticarToken, upload.single('foto'), (req, res) => {
  const id = req.params.id;
  const { nome, email, telefone, senha, curso_id, serie, FK_tipo_usuario_id } = req.body;
  const foto = req.file ? req.file.filename : undefined;

  const updates = [];
  const values = [];

  if (nome !== undefined) { updates.push("nome = ?"); values.push(nome); }
  if (email !== undefined) { updates.push("email = ?"); values.push(email); }
  if (telefone !== undefined) { updates.push("telefone = ?"); values.push(telefone); }
  if (senha !== undefined) { updates.push("senha = ?"); values.push(senha); } // Pode criptografar se quiser
  if (curso_id !== undefined) { updates.push("curso_id = ?"); values.push(curso_id); }
  if (serie !== undefined) { updates.push("serie = ?"); values.push(serie); }
  if (FK_tipo_usuario_id !== undefined) { updates.push("FK_tipo_usuario_id = ?"); values.push(FK_tipo_usuario_id); }
  if (foto !== undefined) { updates.push("foto = ?"); values.push(foto); }

  if (updates.length === 0) return res.status(400).json({ error: "Nenhum campo para atualizar." });

  const sql = `UPDATE usuario SET ${updates.join(", ")} WHERE id = ? AND FK_instituicao_id = ?`;
  values.push(id, req.user.FK_instituicao_id);

  connection.query(sql, values, (err, result) => {
    if (err) {
      console.error("Erro ao atualizar usuário:", err);
      return res.status(500).json({ error: "Erro ao atualizar usuário." });
    }
    if (result.affectedRows === 0) return res.status(404).json({ error: "Usuário não encontrado." });
    res.json({ message: "Usuário atualizado com sucesso!" });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

O código apresentado na figura 117 corresponde à rota de atualização de usuário no sistema BibliONtec, implementada em Node.js com o uso do *framework Express*. Essa rota é responsável por receber as requisições do tipo *PUT* e aplicar modificações nos dados de um usuário já cadastrado no banco de dados. Logo no início, é definida a rota /api/usuarios/:id, na qual o parâmetro: id identifica o usuário que será atualizado. Além disso, são utilizados dois *middlewares*: um para autenticar o *token* de acesso, garantindo que apenas usuários autorizados possam realizar alterações, e outro para permitir o *upload* de uma foto de perfil.

Em seguida, o sistema extrai os dados enviados na requisição, como nome, *e-mail*, telefone, senha, curso, série e tipo de usuário, além de verificar se foi anexada uma foto. Esses valores são organizados em *arrays* que armazenam os campos a serem atualizados e seus respectivos conteúdos. Dessa forma, somente os dados realmente enviados na requisição são modificados, evitando sobrescrever informações desnecessárias.

O código constrói dinamicamente o comando SQL para atualizar os registros na tabela de usuários, utilizando a instrução *UPDATE*. Caso nenhum dado seja informado, o sistema retorna um erro informando que não há campos para atualização. Já no momento da execução, se ocorrer algum problema na comunicação com o banco, é retornado um erro interno, enquanto a ausência de registros correspondentes ao ID informado gera a resposta de “Usuário não encontrado”. Por outro lado, se a operação for concluída com sucesso, o sistema retorna a mensagem confirmando que o usuário foi atualizado.

Assim, essa parte do código garante a integração entre a interface do sistema e o banco de dados, permitindo ao professor ou administrador manter suas informações pessoais sempre atualizadas e assegurando maior consistência e confiabilidade dos dados cadastrados na aplicação

3.4.18 – Indicar livros

Figura 118 - Tela do professor indicar livro



FONTE: AUTORIA PRÓPRIA (2025)

A figura 118 apresenta a interface gráfica de um sistema de indicação de livros, voltado para o contexto educacional. Nela, o professor pode visualizar informações detalhadas sobre um livro, incluindo o título ("Algoritmos e Programação"), o autor (Alexandre Dumas) e uma descrição do conteúdo do livro. O sistema é configurado de forma a oferecer uma interação intuitiva com os alunos ou professores, permitindo que eles verifiquem a disponibilidade do livro. Nesse caso, o livro em questão está marcado como "Disponível", o que indica que os professores podem realizar a reserva do livro.

Além disso, é visível um campo de comentários onde os professores podem compartilhar suas opiniões sobre o livro. Esse componente facilita a interação social dentro da plataforma, permitindo a troca de experiências e opiniões entre os participantes. A presença do botão de indicação de livros sugere que o sistema também possibilita aos professores fazerem sugestões de livros, associando-os a suas respectivas turmas.

Essa interface é alimentada por um *back-end* que utiliza banco de dados para gerenciar o estado de disponibilidade dos livros, as indicações feitas pelos professores e outros aspectos dinâmicos da aplicação.

Figura 119 - Código de indicação do *modal*



```
// --- SISTEMA DE INDICAÇÃO COM MODAL (MULTI TURMAS) ---
const modalTurma = new bootstrap.Modal(document.getElementById('modalTurma'));
let turmasGlobais = [];

if (btnIndicar && isProfessor) {
  btnIndicar.addEventListener("click", async () => {
    if (!usuarioId) return alert("Você precisa estar logado para indicar um livro.");

    try {
      const respTurmas = await fetch('http://localhost:3000/turmas');
      if (!respTurmas.ok) throw new Error("Erro ao buscar turmas");

      const data = await respTurmas.json();

      turmasGlobais = data.turmas_agrupadas.map(turma => {
        const curso = data.cursos.find(c => c.id === turma.curso_id);
        return {
          curso_id: turma.curso_id,
          curso: curso ? curso.curso : "Curso desconhecido",
          serie: turma.serie
        };
      });

      if (!turmasGlobais.length) {
        alert("Nenhuma turma encontrada.");
        return;
      }

      preencherSelectTurmas(turmasGlobais);
      modalTurma.show();

    } catch (err) {
      console.error("Erro ao carregar turmas:", err);
      alert("Erro ao carregar turmas.");
    }
  });
}

function preencherSelectTurmas(turmas) {
  const select = document.getElementById('selectTurmas');
  select.innerHTML = "";

  turmas.forEach(t => {
    const option = document.createElement('option');
    option.value = `${t.curso_id}-${t.serie}`;
    option.textContent = `${t.curso} - ${t.serie}ª série`;
    select.appendChild(option);
  });
}

document.getElementById('btnConfirmarIndicacao').addEventListener('click', async () => {
  const select = document.getElementById('selectTurmas');
  const selecionadas = Array.from(select.selectedOptions).map(opt => opt.value);

  if (!selecionadas.length) {
    alert('Por favor, selecione pelo menos uma turma');
    return;
  }

  try {
    for (const item of selecionadas) {
      const [cursoId, serie] = item.split('-');

      await fetch("http://localhost:3000/indicacoes", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          usuarioId,
          livroId,
          cursoId,
          serie
        })
      });
    }

    btnIndicar.innerHTML = "✓ Já Indicado";
    btnIndicar.disabled = true;
    btnIndicar.style.backgroundColor = "#28a745";
    btnIndicar.style.cursor = "not-allowed";

    modalTurma.hide();
    alert("Livro indicado com sucesso para todas as turmas selecionadas!");
  } catch (err) {
    console.error("Erro ao indicar livro:", err);
    alert("Erro ao salvar indicação.");
  }
});

} catch (err) {
  console.error("Erro no visLivro.js:", err);
  alert("Erro ao carregar detalhes do livro.");
}
});
```

FONTE: AUTORIA PRÓPRIA (2025)

A figura 119 mostra um trecho de código JavaScript utilizado para manipular a interface do sistema de indicação de livros. O código implementa a interação do usuário com o sistema por meio de um *modal*, o qual exibe informações sobre as turmas disponíveis para o processo de indicação.

Utilizando a função *fetch*(), o código realiza requisições assíncronas ao servidor, recuperando informações sobre as turmas associadas aos usuários. O sistema envia uma requisição para o servidor Node.js hospedado localmente, na URL http://localhost:3000/turnos, que retorna um conjunto de dados com as turmas e seus respectivos detalhes. Essas informações são então processadas no lado do cliente (*front-end*), permitindo que o professor selecione a turma para associar à indicação do livro. A utilização do *fetch*() é uma abordagem eficiente para fazer requisições em tempo real, sem a necessidade de recarregar a página, garantindo uma experiência dinâmica e fluida para o usuário.

Esse código JavaScript lida com a manipulação de elementos DOM (*Document Object Model* – Modelo de Documento por Objetos), incluindo a criação e inserção de opções de turmas, atualizando a interface conforme a escolha do usuário. A interação entre o cliente e o servidor é realizada de maneira assíncrona, o que significa que o sistema é capaz de lidar com múltiplas interações simultâneas sem sobrecarregar a interface do usuário.

Além disso, o *script* inclui tratamentos de erro que garantem que, caso a requisição falhe ou não retorne dados, mensagens de alerta sejam exibidas ao professor, mantendo a comunicação clara sobre o estado do sistema. Outro ponto importante é que a modularidade do código facilita a manutenção e futuras atualizações, permitindo que novas funcionalidades sejam integradas sem comprometer a estabilidade da aplicação. A interface é atualizada dinamicamente de acordo com os dados recebidos, garantindo consistência visual e uma experiência mais intuitiva para o usuário. Essa abordagem melhora a eficiência do sistema, permitindo que o processo de indicação seja realizado de forma rápida e confiável, mesmo em situações de múltiplas interações simultâneas.

Figura 120 - Conexão com banco de dados do indicador



```
//================  Indicar livro para uma turma ================
app.post('/indicacoes', (req, res) => {
  const { usuarioId, livroId, cursoId, serie } = req.body;

  // 1. Verificar se usuário é professor
  connection.query(
    'SELECT FK_tipo_usuario_id FROM usuario WHERE id = ?',
    [usuarioId],
    (err, usuarioResults) => {
      if (err) {
        console.error('Erro ao buscar usuário:', err);
        return res.status(500).json({ error: 'Erro interno no servidor' });
      }

      if (usuarioResults.length === 0 || usuarioResults[0].FK_tipo_usuario_id !== 2) {
        return res.status(403).json({ error: 'Apenas professores podem indicar livros' });
      }

      // 2. Verificar se já indicou este livro para a mesma turma
      connection.query(
        `SELECT i.id
         FROM indicacao i
         JOIN indicacao_usuario iu ON i.id = iu.FK_indicacao_id
         WHERE iu.FK_usuario_id = ? AND i.indicacao = ? AND iu.FK_curso_id = ? AND iu.serie = ?`,
        [usuarioId, livroId.toString(), cursoId, serie],
        (err, indicacoesResults) => {
          if (err) {
            console.error('Erro ao verificar indicações:', err);
            return res.status(500).json({ error: 'Erro interno no servidor' });
          }

          if (indicacoesResults.length > 0) {
            return res.status(409).json({ error: 'Você já indicou este livro para esta turma' });
          }

          // 3. Criar indicação
          connection.query(
            'INSERT INTO indicacao (indicacao, FK_instituicao_id) VALUES (?, ?)',
            [livroId.toString(), 1],
            (err, result) => {
              if (err) {
                console.error('Erro ao criar indicação:', err);
                return res.status(500).json({ error: 'Erro interno no servidor' });
              }

              const indicacaoId = result.insertId;

              // 4. Vincular usuário e turma à indicação
              connection.query(
                'INSERT INTO indicacao_usuario (FK_indicacao_id, FK_usuario_id, FK_curso_id, serie) VALUES (?, ?, ?, ?)',
                [indicacaoId, usuarioId, cursoId, serie],
                (err) => {
                  if (err) {
                    console.error('Erro ao vincular indicação:', err);
                    return res.status(500).json({ error: 'Erro interno no servidor' });
                  }

                  res.status(201).json({ message: 'Livro indicado com sucesso para a turma!' });
                }
              );
            }
          );
        }
      );
    }
  );
});
```

FONTE: AUTORIA PRÓPRIA (2025)

A figura 120 apresenta o código Node.js responsável pela verificação e registro de indicações de livros feitas por professores no sistema. O código define uma rota *POST* em /indicacoes, que recebe informações do usuário, do livro e da turma para processar a indicação.

Ao receber a requisição, o sistema primeiro verifica se o usuário é um professor, consultando o banco de dados MySQL pela FK_tipo_usuario_id. Se o

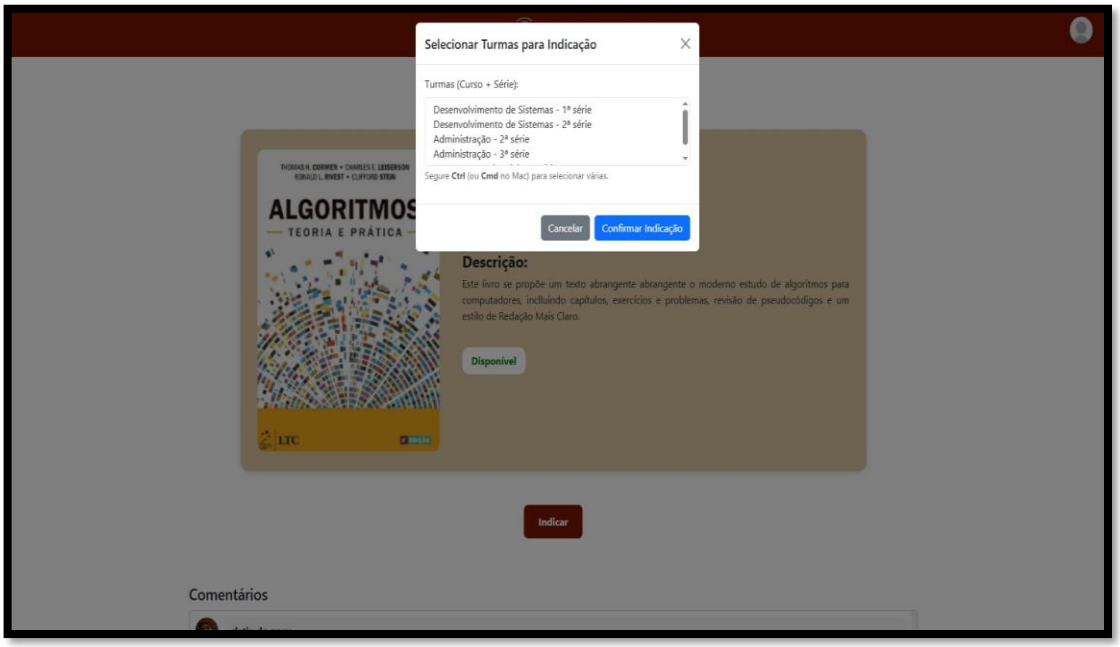
usuário não for professor ou não existir, o servidor retorna uma mensagem de erro e bloqueia a ação.

Em seguida, o código checa se o professor já indicou o mesmo livro para a mesma turma, consultando a relação entre indicações e usuários no banco. Caso a indicação já exista, uma resposta de conflito é enviada, impedindo duplicidade.

Se a indicação ainda não existir, o sistema insere um novo registro na tabela indicacao com o livro selecionado e o ID da instituição. Depois, cria um vínculo entre o usuário e a turma na tabela indicacao_usuario, garantindo que a indicação esteja associada corretamente.

Ao final do processo, se todas as etapas forem bem-sucedidas, o servidor retorna uma mensagem de sucesso para o *front-end*, confirmando que o livro foi indicado para a turma. O uso de chamadas assíncronas permite que a interação entre *front-end* e *back-end* seja fluida e eficiente.

Figura 121 - Tela de *modal* para indicar livros



FONTE: AUTORIA PRÓPRIA (2025)

Na figura 121 vemos uma interface do BibliONtec projetado para a funcionalidade de indicação de livros, destinada a um professor ou responsável pela recomendação de materiais acadêmicos aos alunos. A janela *pop-up* que aparece na tela exibe o título “Selecionar Turmas para Indicação”, o que sugere que o usuário, no

caso um professor, deve escolher para quais turmas o livro será indicado. Dentro da janela, há um campo suspenso (menu de seleção) que permite ao usuário escolher entre diversas opções de cursos e séries, como: Desenvolvimento de Sistemas – 1ª série, Desenvolvimento de Sistemas – 2ª série, Administração – 2ª série e Administração – 3ª série. O menu de seleção é intuitivo, permitindo ao usuário escolher uma ou várias turmas. O sistema instrui o usuário a usar as teclas Ctrl (ou *Command* no Mac) para selecionar múltiplas turmas de uma vez, o que facilita a interação.

Ao lado da lista de turmas, encontra-se uma descrição do livro disponível para indicação, que neste caso é “Algoritmos: Teoria e Prática”. A descrição do livro apresenta uma visão geral do conteúdo abordado, destacando o estudo de algoritmos e suas aplicações, além de incluir capítulos, exercícios e problemas. Abaixo da descrição, há um botão verde rotulado como “Disponível”, indicando que o livro está disponível para ser emprestado.

A janela *pop-up* também exibe dois botões na parte inferior: Cancelar, que permite fechar a janela sem realizar nenhuma ação, e Confirmar Indicação, que confirma a seleção das turmas e a indicação do livro. Essa interface é parte de um processo simples, porém importante, de gestão acadêmica, permitindo aos professores gerenciar materiais de leitura de forma prática e eficiente.

Outro aspecto relevante é que essa funcionalidade ajuda a padronizar a distribuição de conteúdos entre diferentes turmas, evitando que alunos de um mesmo curso tenham acesso desigual a determinados materiais. Além disso, ao centralizar a indicação no sistema, reduz-se a necessidade de comunicações externas, como mensagens ou listas manuais, tornando o processo mais organizado. A interface também contribui para a rastreabilidade das indicações, já que o sistema registra quais livros foram recomendados e para quais turmas, auxiliando no acompanhamento pedagógico. Dessa forma, o recurso não apenas otimiza a rotina dos professores, mas também fortalece o vínculo entre o planejamento acadêmico e o uso da biblioteca digital.

Figura 122 - Código do *modal* indicar livro



```
document.addEventListener("DOMContentLoaded", async () => {
  try {
    // --- Carregar usuário logado ---
    const usuarioData = JSON.parse(localStorage.getItem("usuario"));
    const usuarioId = usuarioData ? usuarioData.id : null;
    const usuarioNome = usuarioData ? usuarioData.nome : "Você";
    const usuarioFoto = usuarioData && usuarioData.foto
      ? `http://localhost:3000/uploads/${usuarioData.foto}`
      : "../img/avatar.jpg";
    const isProfessor = usuarioData && usuarioData.tipo_usuario_id == 2;

    // --- Carregar livro ---
    const livroId = localStorage.getItem("livroSelecionado");
    if (!livroId) throw new Error("Nenhum livro selecionado.");

    const respLivro = await fetch(`http://localhost:3000/livros/${livroId}`);
    if (!respLivro.ok) throw new Error("Erro ao buscar livro.");

    const livro = await respLivro.json();
    console.log("Livro carregado:", livro);

    // Atualizar HTML do livro
    document.querySelector(".livro-detalhe img").src = livro.capa
      ? `http://localhost:3000/uploads/${livro.capa}`
      : "https://via.placeholder.com/320x450";
    document.querySelector(".livro-info h4:nth-of-type(1) + p").textContent = livro.titulo || "Título não disponível";
    document.querySelector(".livro-info h4:nth-of-type(2) + p").textContent = livro.autores || "Autor não informado";
    document.querySelector(".livro-info h4:nth-of-type(3) + p").textContent = livro.sinopse || "Descrição não disponível";

    const disponibilidadeElem = document.querySelector(".livro-info .disponibilidade");
    if (livro.dias_faltando && livro.dias_faltando > 0) {
      disponibilidadeElem.textContent = `Indisponível - Faltam ${livro.dias_faltando} dias`;
      disponibilidadeElem.style.color = "red";
    } else {
      disponibilidadeElem.textContent = "Disponível";
      disponibilidadeElem.style.color = "green";
    }

    // --- SISTEMA DE INDICAÇÃO - VERIFICAÇÃO INICIAL ---
    const btnIndicar = document.getElementById("btnIndicar");
    if (btnIndicar && isProfessor && usuarioId) {
      try {
        const respVerificacao = await fetch(`http://localhost:3000/verificar-indicacao/${usuarioId}/${livroId}`);

        if (respVerificacao.ok) {
          const data = await respVerificacao.json();
          if (data.indicado) {
            btnIndicar.style.display = "none";
            const btnDesindicar = document.getElementById("btnDesindicar");
            btnDesindicar.style.display = "inline-block";

            btnDesindicar.addEventListener("click", async () => {
              if (!confirm("Deseja realmente remover a indicação deste livro?")) return;

              try {
                const resp = await fetch(`http://localhost:3000/indicacoes/${usuarioId}/${livroId}`, {
                  method: "DELETE"
                });

                if (resp.ok) {
                  alert("Indicação removida com sucesso!");
                  btnDesindicar.style.display = "none";
                  btnIndicar.style.display = "inline-block";
                  btnIndicar.disabled = false;
                  btnIndicar.innerHTML = "Indicar";
                  btnIndicar.style.backgroundColor = "#A1600";
                  btnIndicar.style.cursor = "pointer";
                } else {
                  const erro = await resp.text();
                  console.error("Erro ao remover indicação:", erro);
                  alert("Erro ao remover indicação.");
                }
              } catch (err) {
                console.error("Erro de conexão:", err);
                alert("Erro ao tentar remover indicação.");
              }
            });
          }
        } else if (respVerificacao.status === 500) {
          console.warn("Servidor retornou erro 500 na verificação de indicação");
        }
      } catch (err) {
        console.error("Erro ao verificar indicação:", err);
      }
    }
```

FONTE: AUTORIA PRÓPRIA (2025)

A figura 122 exibe um trecho de código JavaScript responsável pela interação entre o *front-end* e o *back-end* de um sistema de indicação de livros. O código é voltado para a parte visual da aplicação, interagindo com os elementos da interface e garantindo que as informações sejam apresentadas de maneira dinâmica ao usuário. O *script* inicia ao carregar o conteúdo da página, capturando as informações do livro armazenadas no *localStorage*, como o título, autor e descrição. Essas informações são então exibidas na interface, com o texto sendo atualizado conforme o conteúdo do livro selecionado.

Um dos pontos centrais do código é a verificação de uma indicação existente. Quando o usuário tenta fazer uma nova indicação, o código faz uma requisição ao *back-end* para confirmar se o livro já foi indicado anteriormente para o mesmo usuário.

Essa verificação é feita com uma chamada *fetch*, que consulta o servidor e retorna a resposta de acordo com o estado da indicação. Se o livro já tiver sido indicado, a interface do usuário é atualizada para mostrar um *feedback* visual, como o destaque de um botão ou a alteração de cores, para informar ao professor sobre o estado da indicação.

Além disso, o código trata de eventos de clique, alterando a aparência dos botões conforme o processo avança. Caso o usuário realize uma indicação corretamente, o sistema mostra uma confirmação visual de sucesso, enquanto, se ocorrer algum erro, um *feedback* de falha é exibido, mantendo a comunicação clara entre o sistema e o usuário. Esse tipo de interação proporciona uma experiência de usuário fluida e intuitiva, essencial para aplicações acadêmicas que exigem uma navegação simples e eficaz.

Outro aspecto relevante é o uso do *localStorage*, que permite armazenar informações diretamente no navegador, sem a necessidade de constantes consultas ao servidor. Essa estratégia melhora o desempenho do sistema, uma vez que reduz o tráfego de dados e agiliza o carregamento da interface. Além disso, garante que mesmo em situações de conexão instável, o usuário consiga visualizar temporariamente os dados já carregados.

O código também incorpora boas práticas de programação assíncrona por meio do uso do *fetch* aliado ao tratamento de promessas. Isso evita que a aplicação fique travada durante a comunicação com o *back-end*, oferecendo uma resposta mais rápida ao usuário. Em caso de falha na requisição, mensagens de erro claras podem ser exibidas, garantindo que o professor entenda o motivo da falha e possa tentar novamente.

Vale destacar ainda que a organização das funções em blocos separados dentro do *script* facilita a manutenção futura do sistema. Essa modularidade permite que outros desenvolvedores compreendam com mais facilidade o fluxo de execução, além de possibilitar a expansão da funcionalidade sem comprometer o restante do código.

Figura 123 - Conexão com banco de dados do *modal*



```
//================ Indicar livro para múltiplas turmas ================

app.post('/indicacoes/multiplas-turmas', (req, res) => {
  const { usuarioId, livroId, turmas } = req.body;

  if (!usuarioId || !livroId || !Array.isArray(turmas) || turmas.length === 0) {
    return res.status(400).json({ error: 'Dados inválidos' });
  }

  connection.query(
    'SELECT FK_tipo_usuario_id FROM usuario WHERE id = ?',
    [usuarioId],
    (err, usuarioResults) => {
      if (err) {
        console.error('Erro ao buscar usuário:', err);
        return res.status(500).json({ error: 'Erro interno no servidor' });
      }

      if (usuarioResults.length === 0 || usuarioResults[0].FK_tipo_usuario_id !== 2) {
        return res.status(403).json({ error: 'Apenas professores podem indicar livros' });
      }

      const indicacoesPromises = turmas.map(turma => {
        return new Promise((resolve, reject) => {
          const { cursoId, serie } = turma;

          connection.query(
            `SELECT i.id
             FROM indicacao i
             JOIN indicacao_usuario iu ON i.id = iu.FK_indicacao_id
             WHERE iu.FK_usuario_id = ? AND i.indicacao = ? AND iu.FK_curso_id = ? AND iu.serie = ?`,
            [usuarioId, livroId.toString(), cursoId, serie],
            (err, results) => {
              if (err) return reject(err);

              if (results.length > 0) {
                return resolve({ cursoId, serie, status: 'já_existe' });
              }

              connection.query(
                'INSERT INTO indicacao (indicacao, FK_instituicao_id) VALUES (?, ?)',
                [livroId.toString(), 1],
                (err, result) => {
                  if (err) return reject(err);

                  const indicacaoId = result.insertId;

                  connection.query(
                    'INSERT INTO indicacao_usuario (FK_indicacao_id, FK_usuario_id, FK_curso_id, serie) VALUES (?, ?, ?, ?)',
                    [indicacaoId, usuarioId, cursoId, serie],
                    (err) => {
                      if (err) return reject(err);
                      resolve({ cursoId, serie, status: 'sucesso' });
                    }
                  );
                }
              );
            }
          );
        });
      });

      // Executar todas as indicações
      Promise.all(indicacoesPromises)
        .then(results => {
          const sucessos = results.filter(r => r.status === 'sucesso');
          const jaExistentes = results.filter(r => r.status === 'já_existe');

          res.status(201).json({
            message: `Indicações processadas: ${sucessos.length} novas, ${jaExistentes.length} já existiam`,
            detalhes: results
          });
        })
        .catch(error => {
          console.error('Erro ao processar indicações:', error);
          res.status(500).json({ error: 'Erro ao processar indicações' });
        });
    }
  );
});
```

FONTE: AUTORIA PRÓPRIA (2025)

A figura 123 apresenta um trecho do código *back-end* desenvolvido utilizando Node.js, responsável por gerenciar a lógica de indicação de livros no sistema. Este

módulo realiza a comunicação com o banco de dados e controla o processo de verificação e inserção de novas indicações feitas pelos professores para diferentes turmas.

O código inicia validando os dados recebidos na requisição, como o identificador do usuário, o identificador do livro e a lista de turmas. Em seguida, é feita uma consulta ao banco de dados para verificar o tipo do usuário, garantindo que apenas usuários com permissão (professores) possam realizar indicações. Caso o usuário não possua o tipo correto, o sistema bloqueia a operação, prevenindo alterações indevidas no acervo acadêmico.

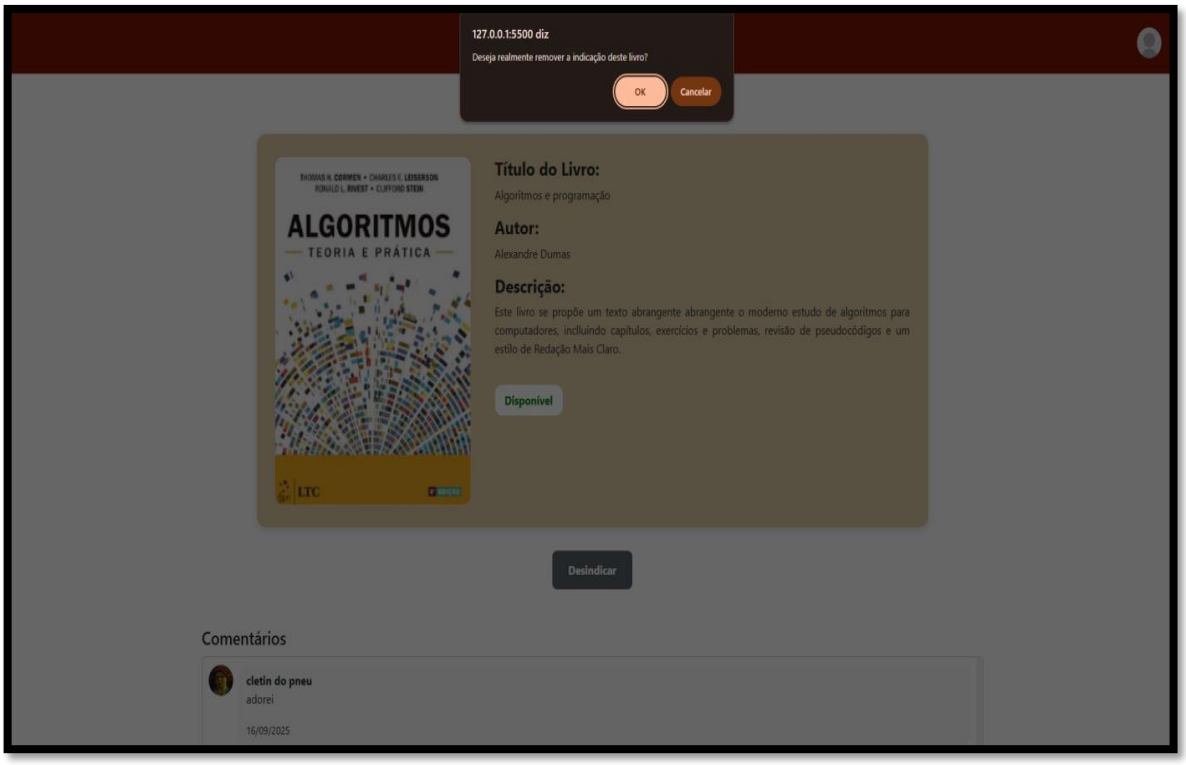
Após a validação do perfil do usuário, o sistema percorre a lista de turmas selecionadas e executa, para cada uma delas, uma verificação no banco de dados para identificar se aquele livro já foi indicado anteriormente para a mesma turma. Essa verificação evita duplicidades nas indicações e mantém o registro organizado e confiável.

Se o livro ainda não tiver sido indicado para a turma, o sistema realiza duas inserções no banco de dados: a primeira na tabela de indicações e a segunda na tabela que relaciona a indicação ao professor e à respectiva turma. Todo esse processo é encapsulado em promessas (*Promises*), o que permite o tratamento assíncrono das operações, evitando bloqueios e garantindo respostas rápidas ao usuário.

Ao final, o sistema retorna uma resposta com o resumo da operação, indicando quantas indicações foram registradas com sucesso e quantas já existiam. A utilização de Node.js, combinada com boas práticas de verificação e inserção, garante a integridade dos dados, a consistência do processo de indicação no sistema e a confiabilidade das informações apresentadas para professores e administradores.

Além disso, a modularidade do código permite que futuras atualizações ou expansões do sistema sejam implementadas de forma mais simples, mantendo a escalabilidade e a manutenção facilitada do *back-end*.

Figura 124 - Tela de desindicar livro



FONTE: AUTORIA PRÓPRIA (2025)

A figura 124 apresentada retrata a interface do BibliONtec, acessada por um usuário com perfil de professor, onde é possível visualizar a indicação de um livro. O livro exibido na tela tem como título “Algoritmos – Teoria e Prática”, de autoria de Alexandre Dumas. A descrição do livro informa que se trata de um texto abrangente voltado ao estudo moderno de algoritmos para computadores, incluindo capítulos, exercícios, problemas, revisão de pseudocódigos e um estilo de redação mais claro. O sistema indica que o livro está disponível por meio de um selo com o texto “Disponível”, posicionado logo abaixo da descrição.

Centralizado na parte inferior do cartão do livro, há um botão com o rótulo “Desindicar”, que permite ao professor remover a indicação feita anteriormente. Ao clicar nesse botão, o sistema exibe uma janela de confirmação no topo da tela, com a mensagem “Deseja realmente remover a indicação deste livro?”, acompanhada de dois botões: “OK” e “Cancelar”. O alerta é gerado a partir do navegador, conforme evidenciado pelo endereço 127.0.0.1:5500, que demonstra que a aplicação está sendo executada localmente em ambiente de desenvolvimento. Esse mecanismo

garante que o professor confirme a ação antes de qualquer alteração no banco de dados, evitando remoções acidentais.

Abaixo da área principal do livro, há uma seção de comentários. Nela, é exibido um comentário feito por um usuário identificado como “cletin do pneu”, que escreveu a mensagem “adorei”, acompanhada da data 16/09/2025. Essa funcionalidade permite que os usuários deixem suas opiniões sobre os livros indicados, agregando valor à experiência de uso da plataforma e promovendo a interação entre docentes e alunos.

Além disso, a interface oferece um *design* responsivo e organizado, facilitando a navegação mesmo em diferentes dispositivos, como tablets ou *notebooks*. A disposição clara dos elementos visuais, juntamente com o uso de cores e destaques, contribui para que o professor identifique rapidamente as funcionalidades disponíveis. A integração de comentários com as indicações também permite um acompanhamento mais detalhado sobre a recepção dos livros pelos usuários, auxiliando no planejamento acadêmico e na recomendação de materiais futuros. A interface ainda conta com mensagens de *feedback* visuais que orientam o usuário durante a interação, reforçando a clareza do sistema.

A interface foi desenvolvida com foco na clareza das ações e boa organização visual, permitindo ao professor indicar ou remover indicações de livros de forma direta, bem como visualizar os comentários de outros usuários. Todo o fluxo de interação exibido está funcionando corretamente, de acordo com as funcionalidades propostas pelo sistema, proporcionando uma experiência intuitiva e eficiente. Além disso, a estrutura adotada favorece futuras atualizações, garantindo que novas funcionalidades possam ser incorporadas sem comprometer a organização e usabilidade da plataforma. Dessa forma, o sistema se mantém flexível e preparado para atender novas demandas conforme evoluir o uso da biblioteca digital. Isso reforça o compromisso em oferecer um ambiente acadêmico mais dinâmico, interativo e alinhado às necessidades de professores e alunos. O código *front-end* segue o mesmo padrão de exclusão visto na figura 122.

Figura 125 - Conexão com banco de dados do desindicar livro



```
//================  Remover Indicação ================
app.delete('/indicacoes/:usuarioId/:livroId', (req, res) => {
  const { usuarioId, livroId } = req.params;

  // Primeiro busca o id da indicação correspondente ao livro
  const sqlBusca = `
    SELECT i.id AS indicacao_id
    FROM indicacao i
    JOIN indicacao_usuario iu ON iu.FK_indicacao_id = i.id
    JOIN livro l ON l.id = ?
    WHERE iu.FK_usuario_id = ?;
  `;

  connection.query(sqlBusca, [livroId, usuarioId], (err, rows) => {
    if (err) {
      console.error(err);
      return res.status(500).json({ error: "Erro ao buscar indicação." });
    }

    if (!rows.length) {
      return res.status(404).json({ error: "Nenhuma indicação encontrada." });
    }

    const indicacaoId = rows[0].indicacao_id;

    const sqlDelete = `DELETE FROM indicacao_usuario WHERE FK_indicacao_id = ? AND FK_usuario_id = ?`;

    connection.query(sqlDelete, [indicacaoId, usuarioId], (err) => {
      if (err) {
        console.error(err);
        return res.status(500).json({ error: "Erro ao remover indicação." });
      }

      res.json({ success: true });
    });
  });
});
```

FONTE: AUTORIA PRÓPRIA (2025)

O trecho de código apresentado na Figura 125 corresponde à implementação da remoção de uma indicação de livro no banco de dados, realizada por meio de uma rota HTTP do tipo *DELETE*. Essa funcionalidade pertence ao *back-end* do sistema, sendo implementada com Node.js e o *framework Express*, utilizando o módulo de conexão com banco de dados MySQL.

A rota definida como *app.delete*('/indicacoes/:usuarioId/:livroId') recebe dois parâmetros na URL: o usuarioId e o livroId. Esses dados são utilizados para localizar no banco de dados a indicação correspondente feita por um usuário (no caso, um professor) a um determinado livro.

O código inicia com a criação de uma instrução SQL que tem como objetivo buscar o identificador da indicação (indicacao_id) a partir da junção de três tabelas: indicacao, indicacao_usuario e livro. Essa busca é feita com base no ID do livro e no ID do usuário. A junção entre as tabelas garante a consistência relacional, verificando se a indicação realmente pertence ao usuário informado e se está associada ao livro correspondente.

Se a consulta retornar resultados, o sistema extrai o ID da indicação e o utiliza para executar uma nova *query* SQL de exclusão, onde o registro é removido da tabela indicacao_usuario, respeitando as chaves estrangeiras FK_indicacao_id e FK_usuario_id. Essa exclusão remove do banco de dados o vínculo entre o usuário e o livro indicado, preservando a integridade das relações já existentes nas demais tabelas do sistema.

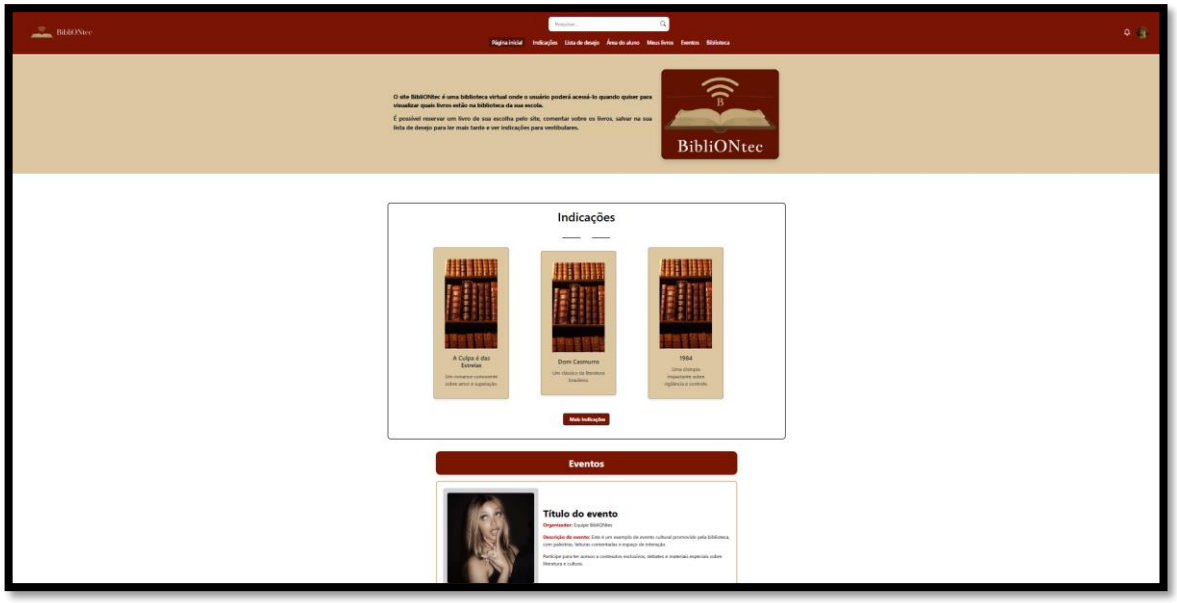
Durante todo o processo, o código trata erros que possam ocorrer tanto na consulta quanto na exclusão, retornando mensagens apropriadas com os códigos de *status* HTTP correspondentes (500 para erro interno e 404 caso a indicação não seja encontrada). Em caso de sucesso, é retornado um objeto JSON com { *success*: *true* }, confirmando que a remoção foi realizada corretamente no banco de dados.

Esse código é a parte responsável por garantir que, ao acionar a funcionalidade de desindicar um livro na interface do sistema, a exclusão da indicação seja efetivamente executada no banco de dados, mantendo a integridade das informações e a fidelidade entre o que é exibido ao usuário e o que está registrado na base de dados. Além disso, a implementação contribui para a experiência do usuário ao evitar inconsistências, já que qualquer exclusão é refletida imediatamente na interface, reforçando a confiabilidade e a transparência do sistema.

Adicionalmente, essa abordagem modular facilita a manutenção do código, permitindo que futuras alterações na lógica de exclusão sejam realizadas de forma simples e sem impacto em outras funcionalidades. A utilização de boas práticas, como o retorno padronizado de mensagens em formato JSON, também contribui para a integração com outras partes do sistema e até mesmo com aplicações externas. Dessa maneira, o processo de remoção não apenas cumpre sua função técnica, mas também reforça a escalabilidade e a robustez da solução desenvolvida.

3.4.19 – Tela Inicial do aluno

Figura 126 - Tela Inicial Aluno



FONTE: AUTORIA PRÓPRIA (2025)

A tela apresentada na figura 126 é a página inicial do sistema BibliONtec, exibida logo após o aluno realizar *login*. Ela funciona como um painel de boas-vindas e de destaques da biblioteca virtual, oferecendo acesso rápido a informações e recursos relevantes.

Na parte superior, há um menu de navegação fixo, que permite ao usuário acessar facilmente as principais seções: Página inicial, Indicações, Lista de desejos, Área do aluno, Meus livros, Eventos e Biblioteca. Também está presente uma barra de pesquisa, para que o aluno encontre rapidamente livros ou conteúdos disponíveis.

Logo abaixo, há um texto introdutório que explica a proposta do sistema: oferecer acesso remoto aos livros, permitir reserva de exemplares, adicionar obras à lista de desejos, comentar e acompanhar recomendações personalizadas, reforçando o caráter interativo e acessível da plataforma.

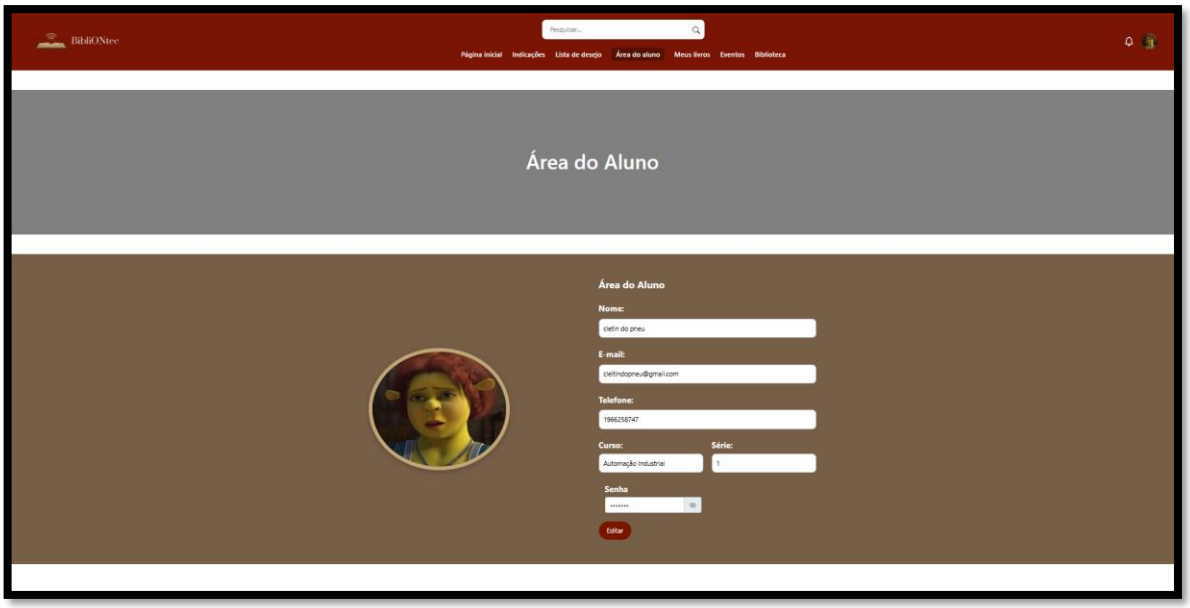
No centro da página, aparece a seção “Indicações”, onde alguns livros são destacados como sugestões de leitura. Cada indicação é apresentada em formato de *card*, contendo a capa da obra, título, autor e uma breve descrição. Isso ajuda os alunos a descobrirem novas leituras de forma prática. Há ainda um botão “Mais Indicações”, que redireciona para a listagem completa das sugestões da biblioteca.

Mais abaixo, está a seção “Eventos”, que divulga atividades culturais e acadêmicas promovidas pela biblioteca. Essa área mostra a imagem de divulgação, título do evento, organizador e uma descrição resumida. Assim, os alunos ficam informados sobre palestras, encontros literários e outras atividades extracurriculares vinculadas à instituição.

Em resumo, essa tela inicial funciona como uma vitrine dinâmica da biblioteca virtual: apresenta indicações de leitura, destaca eventos importantes e dá acesso rápido às funcionalidades do sistema, tornando a experiência do aluno mais interativa, organizada e próxima da realidade de uma biblioteca física.

3.4.20 – Área do aluno

Figura 127 - Tela Área do Aluno



FONTE: AUTORIA PRÓPRIA (2025)

A tela apresentada na figura 127 corresponde à Área do Aluno, espaço onde o usuário pode visualizar e gerenciar seus dados pessoais dentro do sistema BibliONtec. Essa página funciona como um perfil individual, permitindo ao aluno manter suas informações sempre atualizadas e seguras.

Na parte superior, o menu de navegação se mantém fixo, garantindo que o usuário possa alternar facilmente entre as demais seções do sistema, como Página inicial, Indicações, Lista de desejos, Meus livros, Eventos e Biblioteca.

Logo no centro da tela, o título “Área do Aluno” aparece em destaque, reforçando ao usuário que ele se encontra em sua seção de perfil. Abaixo, é exibida uma foto de perfil, que ajuda na identificação visual.

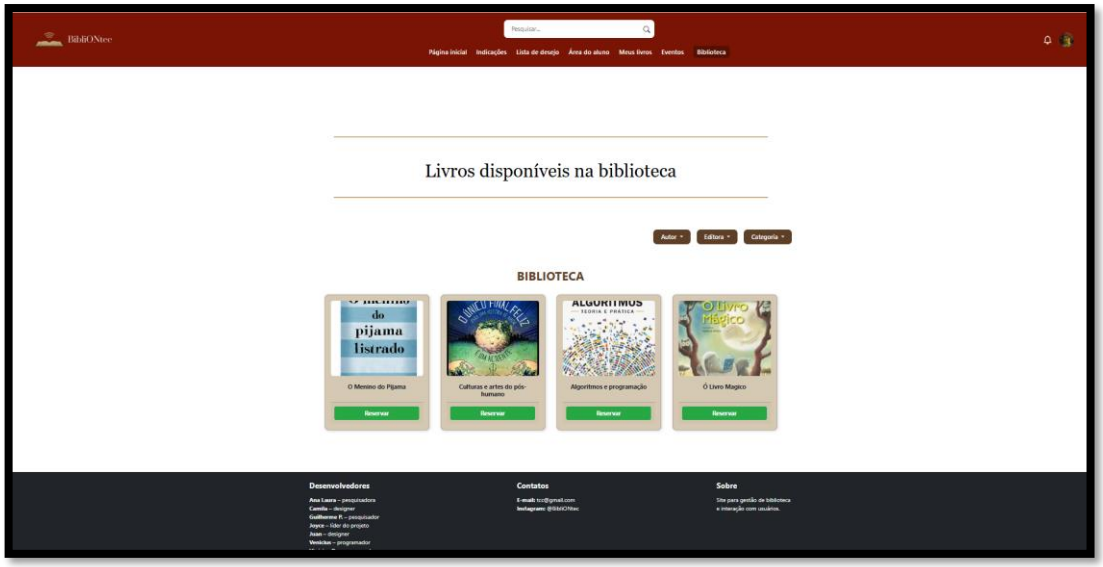
À direita, encontram-se os campos de dados pessoais: nome, *e-mail*, telefone, curso, série e senha. Os campos de telefone e senha são editáveis, permitindo que o aluno atualize as suas informações sempre que necessário, como em caso de mudança de contato ou troca de senha. A presença do botão “Editar” ao final do formulário garante que essas alterações sejam enviadas ao sistema e registradas no banco de dados.

Essa tela também tem um papel de segurança e personalização, pois dá ao aluno maior controle sobre seus próprios dados, evitando inconsistências e permitindo que o sistema mantenha informações atualizadas para comunicação e gestão acadêmica.

Em resumo, a Área do Aluno é um painel de gerenciamento pessoal dentro da biblioteca virtual, onde cada usuário pode administrar suas informações, mantendo seu cadastro sempre correto e atualizado. Isso garante praticidade, segurança e maior autonomia no uso do sistema. O código *front-end* segue o mesmo padrão do apresentado nas figuras 56 e 57 e o *back-end* na figura 117.

3.4.21 – Livros disponíveis aluno

Figura 128 - Tela livros disponíveis



FONTE: AUTORIA PRÓPRIA (2025)

A tela do BibliONtec mostrada na figura 128 apresenta ao usuário todos os livros disponíveis no acervo digital, permitindo uma navegação simples e objetiva. O cabeçalho permanece padronizado, com o menu de navegação fixo e a barra de pesquisa, possibilitando que o aluno acesse rapidamente outras seções do sistema ou busque diretamente por um título específico.

Logo abaixo, em destaque, aparece o título “Livros disponíveis na biblioteca”, seguido de opções de filtro como Autor, Editora e Categoria, que permitem ao usuário refinar a pesquisa de acordo com seus interesses. Essa funcionalidade garante maior organização e rapidez na localização de obras específicas.

Na área principal da tela, é exibida a listagem dos livros cadastrados, dispostos em formato de *cards* com capa, título e um botão de “Reservar”. Esse botão é o recurso central da página, pois possibilita ao aluno reservar o exemplar desejado de maneira prática, sem a necessidade de deslocamento físico até a biblioteca.

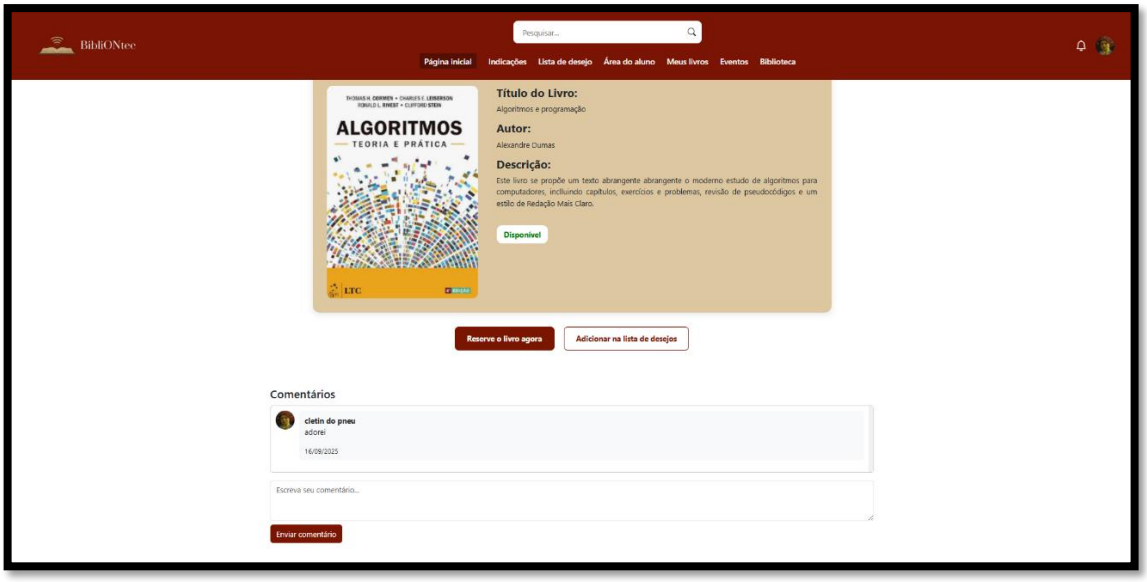
A exibição das capas dos livros também contribui para uma navegação mais intuitiva e atrativa, facilitando a identificação visual das obras. Essa disposição valoriza tanto a apresentação quanto a usabilidade do sistema, estimulando os alunos a explorarem o catálogo.

No rodapé, encontram-se as informações institucionais, como desenvolvedores, contatos e sobre o sistema, reforçando a transparência do projeto e dando credibilidade à aplicação. O código *front-end* e *back-end* seguem o mesmo padrão apresentado nas figuras 83, 84 e 86, com a diferença de não haver alteração e exclusão dos livros.

Além disso, essa tela desempenha um papel estratégico no incentivo ao uso da biblioteca digital, funcionando como um ponto de partida para o engajamento dos alunos. A combinação de praticidade, organização e estética favorece a adoção do sistema, fortalecendo a relação entre a comunidade acadêmica e o acervo digital. Com isso, a biblioteca amplia seu alcance, tornando o acesso ao conhecimento mais democrático e acessível a todos.

3.4.22 - Comentários

Figura 129 - Tela comentários



FONTE: AUTORIA PRÓPRIA (2025)

A tela de Detalhes do Livro, vista na Figura 129, tem como objetivo fornecer ao aluno informações completas sobre um exemplar selecionado no catálogo da biblioteca. Na parte superior, são exibidos a capa do livro, o título, o autor e uma descrição detalhada, permitindo que o usuário compreenda melhor o conteúdo da obra antes de realizar a reserva. Além disso, há um indicador de *status*, que informa se o livro está disponível ou não para empréstimo, trazendo maior clareza no processo de escolha.

Logo abaixo, o sistema disponibiliza dois botões principais: “Reservar o livro agora” e “Adicionar na lista de desejos”. Essa dupla de funcionalidades garante flexibilidade ao usuário, que pode tanto reservar imediatamente o livro quanto apenas incluí-lo em sua lista pessoal para futuras consultas, otimizando sua experiência no ambiente digital e oferecendo diferentes formas de interação com o acervo.

O grande diferencial dessa tela é a seção de comentários, localizada na parte inferior. Nela, os alunos podem registrar suas opiniões sobre o livro, compartilhar impressões e até mesmo trocar recomendações entre colegas. Essa funcionalidade promove um espaço de interação social e colaborativa dentro da plataforma, enriquecendo o uso do sistema e incentivando a troca de experiências literárias entre a comunidade acadêmica.

Além disso, o formulário de envio de comentário é simples e intuitivo, contendo apenas um campo de texto e um botão de envio, o que estimula a participação ativa dos usuários. Para complementar, os comentários já feitos são exibidos de forma organizada, com informações como nome do aluno, data e foto de perfil, garantindo credibilidade e personalização da interação.

Outro ponto importante é que os dados são carregados e atualizados em tempo real, sem necessidade de recarregar a página, o que melhora significativamente a usabilidade. Esse dinamismo aproxima o funcionamento da plataforma de aplicações modernas de redes sociais, tornando a experiência mais agradável e interativa.

Além das funções de reserva e comentários, a tela também pode servir como ferramenta pedagógica, já que os alunos conseguem acompanhar a popularidade e a avaliação das obras entre seus colegas. Isso cria uma espécie de ranking espontâneo, no qual os livros mais comentados e bem avaliados tendem a ganhar destaque dentro da comunidade escolar.

A presença do indicador de *status* em tempo real também contribui para otimizar a gestão do acervo, evitando frustrações no momento da escolha de um livro. Se o exemplar estiver indisponível, o aluno pode optar por adicioná-lo à lista de desejos e ser notificado posteriormente, mantendo a expectativa de leitura.

Outro benefício é a valorização da experiência visual: a capa em destaque, somada às informações organizadas, facilita a decisão do aluno e estimula o engajamento. O design responsivo da tela garante ainda que essas funcionalidades sejam acessíveis tanto em computadores quanto em dispositivos móveis, ampliando a usabilidade.

Do ponto de vista administrativo, essa tela gera dados relevantes para análise, como a quantidade de reservas feitas, os comentários mais frequentes e os livros mais adicionados às listas de desejo. Essas métricas podem auxiliar na tomada de decisão dos gestores da biblioteca, apoiando compras futuras ou identificando obras de maior interesse.

A possibilidade de interação direta com o conteúdo também fortalece o vínculo do aluno com a plataforma, tornando o ato de reservar um livro mais atrativo e

envolvente. Isso vai além da função prática e passa a contribuir para a criação de uma comunidade leitora mais participativa.

Figura 130 - Código comentários



```
// --- Comentários ---
const comentariosContainer = document.getElementById("comentariosContainer");
const novoComentarioInput = document.getElementById("novoComentario");
const btnEnviarComentario = document.getElementById("btnEnviarComentario");

const renderComentarios = (comentarios) => {
  comentariosContainer.innerHTML = "";
  comentarios.forEach(c => {
    const div = document.createElement("div");
    div.classList.add("comentario");
    div.innerHTML = `
      <img src="${c.foto ? `http://localhost:3000/uploads/${c.foto}` : '../img/avatar.jpg'}"
           class="comentario-avatar">
      <div class="comentario-conteudo">
        <strong>${c.nome}</strong>
        <p>${c.comentario}</p>
        <small>${new Date(c.data_comentario).toLocaleDateString('pt-BR')}</small>
      </div>
    `;
    comentariosContainer.appendChild(div);
  });
};

const carregarComentarios = async () => {
  const respComentarios = await fetch(`http://localhost:3000/livros/${livroId}/comentarios`);
  if (!respComentarios.ok) throw new Error("Erro ao buscar comentários");
  const comentarios = await respComentarios.json();
  renderComentarios(comentarios);
};

await carregarComentarios();

if (btnEnviarComentario) {
  btnEnviarComentario.addEventListener("click", async () => {
    if (!usuarioId) return alert("Você precisa estar logado para comentar.");
    const texto = novoComentarioInput.value.trim();
    if (!texto) return alert("Digite um comentário.");

    const resp = await fetch(`http://localhost:3000/livros/${livroId}/comentarios`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ usuarioId, comentario: texto })
    });

    if (!resp.ok) return alert("Erro ao enviar comentário.");
    novoComentarioInput.value = "";
    await carregarComentarios();
  });
}
```

FONTE: AUTORIA PRÓPRIA (2025)

O código apresentado na Figura 130 é responsável por toda a lógica de gerenciamento de comentários nos livros dentro do sistema. Ele se divide em três partes principais: renderização dos comentários já existentes, carregamento dos dados diretamente da API e envio de novos comentários pelos usuários.

Na primeira parte, a função renderComentarios() percorre todos os comentários retornados pelo servidor e monta dinamicamente os elementos que serão exibidos na tela. Cada comentário contém a foto do usuário (ou um avatar padrão, caso não haja foto cadastrada), o nome, o texto do comentário e a data em que foi feito, apresentada no formato brasileiro. Essa etapa garante a organização visual e a clareza das informações mostradas ao aluno, além de permitir que diferentes interações sejam padronizadas de forma consistente.

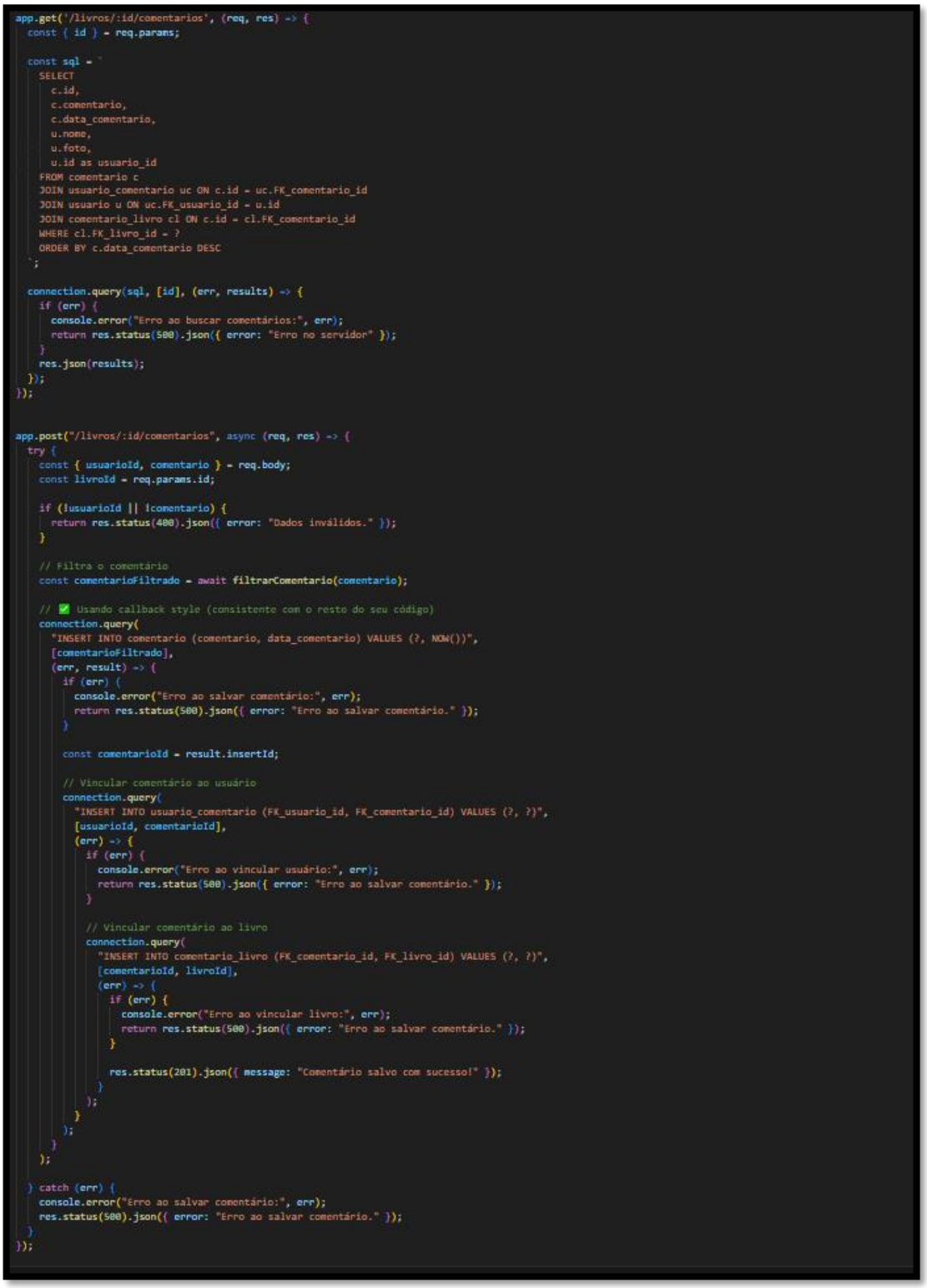
A segunda parte é a função carregarComentarios(), que faz uma requisição ao *back-end* para buscar todos os comentários relacionados a um determinado livro, identificado pelo seu ID. Os dados retornados em formato JSON são enviados para a função de renderização, o que possibilita a atualização automática da área de comentários sempre que a página é carregada. Isso proporciona uma experiência dinâmica ao usuário, sem a necessidade de recarregar manualmente o sistema para visualizar novos conteúdos, reforçando a interatividade da aplicação.

Já a terceira parte corresponde ao envio de novos comentários. Por meio de um evento associado ao botão de envio, o sistema verifica se o aluno está logado e se o campo de texto não está vazio. Caso atenda a esses requisitos, é feita uma requisição *POST* para a API, enviando o ID do usuário e o conteúdo do comentário. Se a operação for bem-sucedida, o campo de texto é limpo e a lista de comentários é atualizada em tempo real, exibindo imediatamente o novo registro. Esse fluxo garante maior fluidez na interação, fortalecendo o engajamento dos alunos com a plataforma.

Além disso, a implementação considera boas práticas de usabilidade, como *feedback* imediato das ações do usuário e a organização cronológica dos comentários, o que contribui para um ambiente colaborativo. Essa funcionalidade pode, inclusive, servir como um espaço de troca de ideias entre os estudantes, transformando o sistema não apenas em um gerenciador de livros, mas também em uma ferramenta de interação acadêmica. Dessa forma, o gerenciamento de

comentários reforça a proposta pedagógica da plataforma, estimulando a participação ativa e a construção coletiva de conhecimento.

Figura 131 - Conexão com banco de dados comentários



```
app.get('/livros/:id/comentarios', (req, res) => {
  const { id } = req.params;

  const sql = `
    SELECT
      c.id,
      c.comentario,
      c.data_comentario,
      u.nome,
      u.foto,
      u.id as usuario_id
    FROM comentario c
    JOIN usuario_comentario uc ON c.id = uc.FK_comentario_id
    JOIN usuario u ON uc.FK_usuario_id = u.id
    JOIN comentario_livro cl ON c.id = cl.FK_comentario_id
    WHERE cl.FK_livro_id = ?
    ORDER BY c.data_comentario DESC
  `;

  connection.query(sql, [id], (err, results) => {
    if (err) {
      console.error("Erro ao buscar comentários:", err);
      return res.status(500).json({ error: "Erro no servidor" });
    }
    res.json(results);
  });
});


app.post("/livros/:id/comentarios", async (req, res) => {
  try {
    const { usuarioId, comentario } = req.body;
    const livroId = req.params.id;

    if (!usuarioId || !comentario) {
      return res.status(400).json({ error: "Dados inválidos." });
    }

    // Filtra o comentário
    const comentarioFiltrado = await filtrarComentario(comentario);

    // ✅ Usando callback style (consistente com o resto do seu código)
    connection.query(
      "INSERT INTO comentario (comentario, data_comentario) VALUES (?, NOW())",
      [comentarioFiltrado],
      (err, result) => {
        if (err) {
          console.error("Erro ao salvar comentário:", err);
          return res.status(500).json({ error: "Erro ao salvar comentário." });
        }

        const comentarioId = result.insertId;

        // Vincular comentário ao usuário
        connection.query(
          "INSERT INTO usuario_comentario (FK_usuario_id, FK_comentario_id) VALUES (?, ?)",
          [usuarioId, comentarioId],
          (err) => {
            if (err) {
              console.error("Erro ao vincular usuário:", err);
              return res.status(500).json({ error: "Erro ao salvar comentário." });
            }

            // Vincular comentário ao livro
            connection.query(
              "INSERT INTO comentario_livro (FK_comentario_id, FK_livro_id) VALUES (?, ?)",
              [comentarioId, livroId],
              (err) => {
                if (err) {
                  console.error("Erro ao vincular livro:", err);
                  return res.status(500).json({ error: "Erro ao salvar comentário." });
                }

                res.status(201).json({ message: "Comentário salvo com sucesso!" });
              }
            );
          }
        );
      }
    );

  } catch (err) {
    console.error("Erro ao salvar comentário:", err);
    res.status(500).json({ error: "Erro ao salvar comentário." });
  }
});
```

FONTE: AUTORIA PRÓPRIA (2025)

O código apresentado na figura 131 é responsável por implementar a lógica de comunicação entre a aplicação e o banco de dados no que diz respeito aos comentários feitos nos livros. Ele define duas rotas principais: uma para listar os comentários existentes e outra para registrar novos comentários.

Na primeira parte, temos a rota *GET* /livros/:id/comentarios, que recebe o identificador do livro e busca no banco todos os comentários associados a ele. A consulta SQL utilizada faz junções entre as tabelas de comentários, usuários e livros, de forma a retornar informações completas, como o texto do comentário, a data em que foi feito, além dos dados do usuário responsável (nome e foto). Os resultados são enviados em formato JSON para o *front-end*, que por sua vez renderiza os comentários na tela. Caso ocorra algum erro durante a consulta, o servidor retorna uma mensagem de falha e um *status* de erro 500.

Na segunda parte, temos a rota *POST* /livros/:id/comentarios, que recebe os dados enviados pelo usuário (ID do usuário e texto do comentário). Antes de salvar no banco, o sistema faz uma verificação para garantir que as informações não estejam vazias. O comentário é então filtrado, salvo na tabela principal junto com a data de criação e vinculado ao usuário e ao livro correspondente por meio de tabelas de relacionamento. Esse processo assegura a integridade das informações e o correto relacionamento entre as entidades do sistema.

Se todas as etapas ocorrerem com sucesso, a API retorna uma mensagem confirmando que o comentário foi salvo. Caso contrário, mensagens de erro são geradas e enviadas ao *front-end*, garantindo clareza ao administrador sobre eventuais problemas no cadastro.

CONSIDERAÇÕES FINAIS

Este trabalho teve como mote a modernização da biblioteca escolar da ETEC, diante da constatação de que o modelo atual apresenta limitações significativas para alunos e bibliotecários – a ausência de sistema informatizado dificulta o controle de acervo, provoca atrasos nas devoluções e impede acesso rápido às informações. Essa problemática realça a importância do tema, pois uma biblioteca eficiente é crucial para promover o hábito de leitura e apoiar o processo educativo. A análise dos dados obtidos por entrevistas e questionários confirmou a existência de uma demanda real por soluções digitais: alunos, professores e bibliotecários manifestaram interesse em uma plataforma *online* que torne a biblioteca mais ágil, organizada e acessível. Assim, a justificativa do projeto – contribuir para uma experiência de uso da biblioteca mais moderna e eficiente – se mostrou pertinente e de grande relevância ao contexto escolar.

O objetivo geral do projeto foi desenvolver o sistema BibliONtec com vistas a facilitar o gerenciamento do acervo e agilizar os processos internos da biblioteca, disponibilizando consulta *online* aos usuários. Este objetivo foi cumprido plenamente: a plataforma criada permite que alunos e professores consultem o catálogo virtual de forma intuitiva, enquanto a equipe da biblioteca dispõe de um painel administrativo integrado a um banco de dados que engloba o cadastro de exemplares, o controle de empréstimos, o monitoramento de prazos e a geração de relatórios. Em suma, o sistema desenvolvido atendeu ao propósito inicial de tornar a gestão bibliotecária mais rápida, transparente e eficaz, conforme previsto na proposta.

Quanto aos objetivos específicos, foram definidos requisitos funcionais e não-funcionais diversos, vinculados às tarefas da biblioteca escolar. Dentre eles estavam a implementação de filtros de busca inteligentes no catálogo *online* e a oferta de uma interface responsiva para acesso em diferentes dispositivos. Além disso, buscou-se desenvolver módulos para cadastro de usuários e funcionários, reserva de exemplares (com fila de espera), e edição de conteúdo do acervo. Na prática, alguns objetivos não foram alcançados na realização do projeto, como a realização de *backups* ou o sistema de notificações, além da geração de relatórios.

A hipótese norteadora do estudo era de que a informatização dos processos bibliotecários melhoraria a organização do acervo e a experiência dos usuários. A execução do projeto confirmou essa hipótese: o sistema BibliONtec endereça os principais entraves identificados no funcionamento atual da biblioteca e incorpora melhorias apontadas na pesquisa preliminar. Em particular, a resposta positiva dos usuários consultados sobre funcionalidades como reservas *online* e consultas remotas corrobora a hipótese inicial. Por conseguinte, conclui-se que a implementação de uma plataforma digital de gestão bibliotecária se mostrou eficaz para suprir as necessidades mapeadas e para cumprir o papel proposto de modernizar o ambiente da biblioteca.

Em resposta ao problema de pesquisa, ou seja, à dificuldade de gerir acervos de forma manual e ineficiente, constatou-se que o BibliONtec o resolve oferecendo automação e acesso remoto. O sistema permite eliminar práticas arcaicas (como controle manual em planilhas) e substituí-las por processos digitais integrados. Por exemplo, o controle de empréstimos agora é realizado em tempo real no próprio sistema, reduzindo erros administrativos, e os usuários podem verificar *online* a disponibilidade dos livros antes de se deslocarem. Dessa forma, o projeto atendeu o problema central levantado, propondo uma solução viável que melhora a eficiência da biblioteca escolar.

A metodologia adotada integrou atividades de pesquisa, análise e desenvolvimento de *software*. Inicialmente, foram realizados levantamentos de requisitos por meio de entrevistas com diretores, bibliotecários e professores, além de questionários aplicados aos alunos. Esses dados orientaram a elaboração de modelos e diagramas (como casos de uso e fluxogramas BPMN) e a criação de protótipos de telas (via ferramentas de *design*). Posteriormente, procedeu-se à implementação tecnológica do sistema utilizando linguagens e ferramentas *web* (JavaScript, Node.js, HTML/CSS e banco de dados MySQL). Em síntese, a abordagem seguiu etapas clássicas de engenharia de *software* – definição de requisitos, modelagem, codificação e testes – alinhando-se às práticas recomendadas para projetos desta natureza.

Durante a execução do projeto, foram enfrentadas limitações que restringiram o escopo de implementação. Em termos de recursos humanos, a equipe possuía

conhecimento intermediário em certas tecnologias, especialmente no que tange a estilização avançada em CSS e ao uso de *frameworks* Node.js (*Express*), demandando tempo extra de aprendizado. Essa limitação técnica exigiu simplificações pontuais no *design* visual e no tratamento de rotas do *back-end*. Além disso, a implantação do sistema em ambiente de produção (hospedagem na *web*) trouxe desafios inéditos: a equipe precisou aprofundar-se em conceitos de hospedagem e configuração de servidor, o que consumiu parte do prazo disponível. Soma-se a isso o fato de que o projeto foi realizado no período definido pelo cronograma do curso, limitando o aprofundamento em funcionalidades extra (por exemplo, recursos avançados de acessibilidade ou integração biométrica). Tais restrições de tempo e recursos devem ser consideradas ao avaliar os resultados alcançados.

Por fim, sugerem-se algumas recomendações para trabalhos futuros e evolução do sistema. Em primeiro lugar, recomenda-se a expansão do catálogo digital para incluir novos formatos e categorias de material (como quadrinhos, periódicos e obras digitais), além de enriquecer o vocabulário controlado e a classificação automática do acervo, conforme apontado nas entrevistas. O desenvolvimento de uma versão móvel dedicada (APP) ou a otimização do *site* para uso em *smartphones* podem aumentar ainda mais a acessibilidade. Também é aconselhável aprimorar as funcionalidades de comunicação, como implementar notificações via *e-mail*/SMS completamente automatizadas e integrar sistemas de recomendação de livros com base no histórico de usuário. Por fim, recomenda-se realizar testes de usabilidade com grupos de alunos e funcionários para validar e refinar a interface, bem como estabelecer um plano de manutenção contínua (atualizações de segurança, *backups* regulares, etc.). Essas ações assegurariam que o BibliONtec evolua de acordo com as necessidades futuras da comunidade escolar.

Em suma, as considerações finais confirmam que o desenvolvimento do BibliONtec alcançou seu propósito principal e contribuiu para a modernização do gerenciamento da biblioteca, ao mesmo tempo em que evidencia pontos de melhoria para continuidade do projeto. As conquistas obtidas indicam que sistemas educacionais similares podem se beneficiar de soluções digitais focadas em eficiência operacional e experiência do usuário, conforme demonstrado neste trabalho.

REFERÊNCIAS

AGÊNCIA FOLLOW. O que é mobile? Conheça mais sobre a tecnologia. Blog Agência Follow, 05 jan. 2024. Disponível em: <https://blog.agenciafollow.com.br/o-que-e-mobile-conheca-mais-sobre-a-tecnologia/>. Acesso em: 10 abr. 2025.

ALVES, Igor. Diferenças entre modelo lógico e modelo físico. Alura, 25 jul. 2024. Disponível em: <https://www.alura.com.br/artigos/diferencas-modelo-logico-modelo-fisico>. Acesso em: 10 abr. 2025.

AMAZON. O que é JavaScript? AWS, 2024. Disponível em: <https://aws.amazon.com/pt/what-is/javascript/>. Acesso em: 10 abr. 2025.

ANDREI L. O que é GitHub, para que serve e como usar. Hostinger, 05 jul. 2024. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-github>. Acesso em: 10 abr. 2025.

APACHE FRIENDS. XAMPP - Download. Apache Friends, 2024. Disponível em: <https://www.apachefriends.org/pt_br/download.html.> Acesso em: 10 abr. 2025.

AWARI. A história do Figma: a evolução da ferramenta de design colaborativo. Awari, 08 ago. 2023. Disponível em: <https://awari.com.br/a-historia-do-figma-a-evolucao-da-ferramenta-de-onlin-colaborativo/>. Acesso em: 10 abr. 2025.

BRASIL ESCOLA. Navegador. Brasil Escola, 2022. Disponível em: <https://m.brasilescola.uol.com.br/informatica/navegador.htm>. Acesso em: 10 abr. 2025.

CALVALCANTI, Laís. VSCode: extensões mais usadas. Alura, 18 set. 2023. Disponível em: <https://www.alura.com.br/artigos/vscode-extensoes-mais-usadas>. Acesso em: 10 abr. 2025.

CANDIDO, H. Carlos. Projeto brModelo. Sourceforge, 10 mai. 2017. Disponível em: <https://sourceforge.net/p/brmodelo/wiki/Projeto%20brModelo/>. Acesso em: 10 abr. 2025.

CANVA. Canva: 10 year timeline. Canva, 25 ago. 2023. Disponível em: <https://www.canva.com/pt_br/midia/novidades/canva-10-year-timeline>. Acesso em: 10 abr. 2025.

CARDOSO, Rodrigo. Servidor Web. Locaweb, 30 jul. 2024. Disponível em: <https://www.locaweb.com.br/blog/temas/codigo-aberto/servidor-web/>. Acesso em: 10 abr. 2025.

CARLOS E. O que é JavaScript? Hostinger, 09 abr. 2025. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-javascript#O_Que_e_JavaScript>. Acesso em: 10 abr. 2025.

CARLOS E. O que é PHP? Guia básico. Hostinger, 12 jan. 2024. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-php-guia-basico>. Acesso em: 10 abr. 2025.

CONTROLE.NET. O que são servidores? Controle.net, 2022. Disponível em: <https://www.controle.net/faq/o-que-sao-servidores>. Acesso em: 10 abr. 2025.

CURVELO, Rakky. Evolução da web. Hubspot Brasil, 03 nov. 2022. Disponível em: <https://br.hubspot.com/blog/marketing/evolucao-web>. Acesso em: 10 abr. 2025.

FERNANDO, Lucas. Desvendando o mundo do CSS. Carreira em TI, 05 fev. 2024. Disponível em: <https://carreiraemti.com.br/blog/desvendando-o-mundo-do-css/>. Acesso em: 10 abr. 2025.

FERRONI, Caroline. O que é mobile? Tecnoseep, 19 mai. 2021. Disponível em: <https://blog.tecnospeed.com.br/o-que-e-mobile/>. Acesso em: 10 abr. 2025.

FIGMA. Sobre o Figma. Figma, 2012. Disponível em: <https://www.figma.com/pt-br/about/>. Acesso em: 10 abr. 2025.

FIGMA. Config 2024 recap. Figma Blog, 26 jun. 2024. Disponível em: <https://www.figma.com/blog/config-2024-recap/>. Acesso em: 10 abr. 2025.

FIGMA. O que é o Figma. Figma Help Center, s.d. Disponível em: <https://help.figma.com/hc/pt-br/articles/14563969806359-O-que-é-o-Figma. Acesso em: 10 abr. 2025.

GITHUB DOCS. Sobre o GitHub e o Git. GitHub Docs, 10 abr. 2025. Disponível em: <https://docs.github.com/pt/get-started/start-your-journey/about-github-and-git>.Acesso em: 10 abr. 2025.

GOBI, Eduardo. XAMPP. Wikipédia, 17 nov. 2015. Disponível em: <https://pt.m.wikipedia.org/wiki/XAMPP>. Acesso em: 10 abr. 2025.

GOMES, Henrique. Modelo lógico de banco de dados: uma visão abrangente. Federal Cubatão, 30 abr. 2024. Disponível em: <https://federalcubatao.com.br/banco-de-dados/modelo-logico-de-banco-de-dados-uma-visao-abrangente/>. Acesso em: 10 abr. 2025.

GOOGLE CLOUD. MySQL no Google Cloud. Google Cloud, s.d. Disponível em: <https://cloud.google.com/mysql?hl=pt-BR>. Acesso em: 10 abr. 2025.

HANASHIRO, Marcelo. Evolução do JavaScript. Locaweb, 12 mar. 2024. Disponível em: <https://www.locaweb.com.br/blog/temas/codigo-aberto/evolucao-do-javascript/. Acesso em: 10 abr. 2025.

HIGA, Paulo. O que é XAMPP e para que serve? Techtudo, 27 fev. 2012. Disponível em: <https://www.techtudo.com.br/google/amp/noticias/2012/02/o-que-e-xampp-e-para-que-serve.ghtml>. Acesso em: 10 abr. 2025.

INFORTREND. Qual é a diferença entre um servidor local e remoto? Infortrend, 2025. Disponível em: <https://www.infortrend.com.br/post/qual-e-a-diferenca-entre-um-servidor-local-e-remoto>. Acesso em: 10 abr. 2025.

KBR TEC. Site dinâmico. KBR Tec, 11 set. 2019. Disponível em: <https://www.kbrtec.com.br/blog/site-dinamico/>. Acesso em: 10 abr. 2025.

MARCIO. Evolução dos servidores. Msserver, 30 set. 2022. Disponível em: <https://msserver.com.br/evolucao-dos-servidores/>. Acesso em: 10 abr. 2025.

MARIADB. Datas de lançamento do MariaDB. MariaDB, s.d. Disponível em: <https://mariadb.com/kb/en/mariadb-server-release-dates/. Acesso em: 10 abr. 2025.

MARQUES, Rafael. O que é HTML? Entenda de forma descomplicada. Homehost, 13 jan. 2024. Disponível em: <https://www.homehost.com.br/blog/tutoriais/o-que-e-html/>. Acesso em: 10 abr. 2025.

MOBILLS. Saiba o que é Canva e como funciona este aplicativo. Mobills, 22 abr. 2023. Disponível em: <https://www.mobills.com.br/blog/o-que-e/saiba-o-que-e-canva-e-como-funciona-este-aplicativo/>. Acesso em: 10 abr. 2025.

NAPOLEON. O que é local server? Napoleon, 16 ago. 2024. Disponível em: <https://napoleon.com.br/glossario/o-que-e-local-server/. Acesso em: 10 abr. 2025.

NETO, Renato. Site estático: o que é. Hostinger, 23 mai. 2024. Disponível em: <https://www.hostinger.com.br/tutoriais/site-estatico>. Acesso em: 10 abr. 2025.

O NOVO MERCADO. História do Canva. O Novo Mercado, s.d. Disponível em: <https://onovomercado.com/blog/historia-canva/>. Acesso em: 10 abr. 2025.

ORACLE. O que é banco de dados? Oracle, 24 nov. 2020. Disponível em: <https://www.oracle.com/br/database/what-is-database/>. Acesso em: 10 abr. 2025.

ORACLE. O que é MySQL? Oracle, 29 ago. 2024. Disponível em: <https://www.oracle.com/br/mysql/what-is-mysql/>. Acesso em: 10 abr. 2025.

PACIEVITCH, Yuri. Conheça o HTML. InfoEscola, s.d. Disponível em: <https://www.infoescola.com/informatica/html/>. Acesso em: 10 abr. 2025.

PROFESSOR JOSÉ HUMBERTO. O que é web? Entenda o conceito e sua importância. Professor José Humberto, 2023. Disponível em: <https://professorjosehumberto.com.br/glossario/o-que-e-web-entenda-o-conceito-e-sua-importancia/>. Acesso em: 10 abr. 2025.

PURE STORAGE. O que é MariaDB? Pure Storage, s.d. Disponível em: <https://www.purestorage.com/br/knowledge/what-is-mariadb.html>. Acesso em: 10 abr. 2025.

REGINALDO. Conhecendo o brModelo. Mundo da Computação Integral, 20 nov. 2021. Disponível em: <https://mundodacomputacaointegral.blogspot.com/2021/11/conhecendo-o-brmodelo.html>. Acesso em: 10 abr. 2025.

REIS, Albery. Evolução do CSS. Dio.me, 23 dez. 2023. Disponível em: <https://www.dio.me/articles/evolucao-do-css>. Acesso em: 10 abr. 2025.

REMESSA ONLINE. Visual Studio Code: confira as principais funções da ferramenta. Remessa Online, 26 out. 2021. Disponível em: <https://www.remessaonline.com.br/blog/visual-studio-code-confira-as-principais-funcoes-da-ferramenta/>. Acesso em: 10 abr. 2025.

SALES, Janderson. Modelo conceitual de banco de dados. Guia de Hospedagem, 28 mar. 2024. Disponível em: <https://guiadehospedagem.com.br/banco-de-dados-modelo-conceitual/>. Acesso em: 10 abr. 2025.

SANTANA, Bruno. O que é CSS? Guia básico de CSS. Hostinger, 08 abr. 2025. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-css-guia-basico-de-css>. Acesso em: 10 abr. 2025.

SPACE PROGRAMMER. Introdução ao modelo de dados e seus níveis de abstração. Space Programmer, s.d. Disponível em: <https://spaceprogrammer.com/bd/introducao-ao-modelo-de-dados-e-seus-niveis-de-abstracao/>. Acesso em: 10 abr. 2025.

UFRN. História dos bancos de dados. UFRN, s.d. Disponível em: <https://materialpublic.imd.ufrn.br/curso/disciplina/4/56/1/3>. Acesso em: 10 abr. 2025.

VIEIRA, D. Conheça o HTML. HostGator, 24 ago. 2020. Disponível em: <https://www.hostgator.com.br/blog/conheca-o-html/>. Acesso em: 10 abr. 2025.

WIX.COM. Saiba tudo sobre sites dinâmicos e estáticos. Wix.com, 24 mar. 2022. Disponível em: <https://pt.wix.com/blog/guia-sites-dinamicos-e-estaticos>. Acesso em: 10 abr. 2025.

EXPERT EM TI. O que é XAPP? Expertemti, 14 mai. 2024. Disponível em: <https://expertemti.com.br/glossario/o-que-e-xampp/>. Acesso em: 11 abr. 2025.

CIMATEC JR. Como fazer um estudo de viabilidade. CIMATEC Jr, s.d. Disponível em: <https://cimatecjr.com.br/como-fazer-um-estudo-de-viabilidade>. Acesso em: 02 mai. 2025.

TÁ CONTRATADO. Viabilidade operacional: importância e análise. Tá Contratado, 30 jan. 2025. Disponível em: <https://tacontratado.com.br/glossario/viabilidade-operacional-importancia-e-analise/>. Acesso em: 02 mai. 2025.

MARTINS, Julia. Estudo de viabilidade: o que é e como fazer um. Asana, 13 mai. 2025. Disponível em: <https://asana.com/pt/resources/feasibility-study>. Acesso em: 20 mai. 2025.

CODEVISUALSTUDIO. Visual Studio Code – Code editing. Redefined. CodeVisualStudio, s.d. Disponível em: <https://code.visualstudio.com/>. Acesso em: 20 mai. 2025.

APACHEFRIENDS. About XAMPP. Apache Friends, s.d. Disponível em: <https://www.apachefriends.org/pt_br/about.html>. Acesso em: 20 mai. 2025.

FIGMA. Pricing. Figma, s.d. Disponível em: <https://www.figma.com/pt-br/pricing/>. Acesso em: 20 mai. 2025.

CANVA. Sobre o Canva. Canva, s.d. Disponível em: <https://www.canva.com/pt_br/about/. Acesso em: 20 mai. 2025.

BRUNO, S. O que é SSL, como funciona e como ter no seu site: guia completo. Hostinger, 02 abr. 2025. Disponível em: <https://www.hostinger.com/br/tutoriais/o-que-e-ssl>. Acesso em: 20 mai. 2025.

ANA, P. Como instalar SSL grátis e vitalício na Hostinger. Hostinger, 01 out. 2024. Disponível em: <https://www.hostinger.com/br/tutoriais/como-instalar-ssl>. Acesso em: 20 mai. 2025.

RECLAMEAQUI. Página da empresa Hostinger no Reclame Aqui. Reclame Aqui, s.d. Disponível em: <https://www.reclameaqui.com.br/empresa/hostinger/>. Acesso em: 20 mai. 2025.

MOULIÈRES, Elodie. O cliente: definição desse importante praticante na empresa. Appvizer, 26 out. 2024. Disponível em: <https://www.appvizer.com.br/revista/vendas-relacao-cliente/crm/definicao-de-cliente-o-ativo-mais-importante-da-empresa>. Acesso em: 04 jun. 2025.

ZENDESK. Atendimento ao cliente interno e externo: boas práticas, técnicas e +. Zendesk, 08 mar. 2024. Disponível em: <https://www.zendesk.com.br/blog/atendimento-ao-cliente-interno-e-externo/>. Acesso em: 04 jun. 2025.

SILVA, Douglas. O que são clientes ativos e inativos? Quais as diferenças. Zendesk, s.d. Disponível em: <https://www.zendesk.com.br/blog/clientes-ativos-e-inativos/>. Acesso em: 04 jun. 2025.

TOTVS. Como agir com clientes insatisfeitos e melhorar sua prestação de serviços. TOTVS, 11 out. 2023. Disponível em: <https://www.totvs.com/blog/gestao-de-servicos/cliente-insatisfeito/#:~:text=Enquanto%20um%20cliente%20satisfeito%20%C3%A9,a%20reputa%C3%A7%C3%A3o%20da%20sua%20companhia>. Acesso em: 04 jun. 2025.

FONSECA, Pedro. Você sabe o que é concorrência? Conheça os tipos e como identificar. Bagy, 2022. Disponível em: <https://bagy.com.br/blog/o-que-e-concorrencia/>. Acesso em: 15 jun. 2025.

BENNETT, Thomas. O que são concorrentes indiretos versus concorrentes diretos. Priceva, 04 jun. 2024. Disponível em: <https://priceva.com/pt/blog/concorrentes-diretos-e-indiretos>. Acesso em: 15 jun. 2025.

KOHACOMMUNITY. Sobre Koha. Koha community, s.d. Disponível em: <https://koha-community.org/about/>. Acesso em: 15 jun. 2025.

ORGANIZADATOS. Koha ILS Host. Organizadatos, s.d. Disponível em: <https://www.organizadatos.com/8-koha/6-koha-ils-hosting-en.html?utm>. Acesso em: 15 jun. 2025.

STERNHOST. Enterprise KOHA Hosting. Sternhost, s.d. Disponível em: <https://sternhost.com/koha/?utm_>. Acesso em: 15 jun. 2025.

BIBLIVRE. O que é o BIBLIVRE. Biblivre, s.d. Disponível em: <https://biblivre.org.br/index.php/sobre-biblivre>. Acesso em: 15 jun. 2025.

BIBLIVRECLOUD. Biblioteca livre. Estável e em qualquer lugar. Biblivrecloud, 2022. Disponível em: <https://biblivre.org.br/index.php/sobre-biblivre>. Acesso em: 15 jun. 2025.

WEINSTEIN, Lior. Catalogue seus livros e gerencie sua biblioteca com o Excel. Cogniview, 18 abr. 2013. Disponível em: <https://www.cogniview.com/blog/catalog-your-books-and-manage-your-library-with-excel/>. Acesso em: 15 jun. 2025.

SARAIVAEDUCACAO. Guia completo dos direitos dos contratos. Saraiva Educação, 23 jun. 2023. Disponível em: <https://conteudo.saraivaeducacao.com.br/juridico/contratos/#:~:text=Os%20contratos%20s%C3%A3o%20regidos%20por%20um%20conjunto%20de%20normas%20legais,relacionadas%20a%20diferentes%20%C3%A1reas%20contratuais>. Acesso em: 15 jun. 2025.

BRASIL. Lei n.º 13.709, de 14 de agosto de 2018. Lei Geral de Proteção de Dados Pessoais (LGPD). Diário Oficial da União, Brasília, 15 ago. 2018. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm>. Acesso em: 16 jun. 2025.

ROSA, Carina Tavares. Entidades privadas têm o dever legal de prestar informações, assim como órgãos públicos? Jusbrasil, 03 ago. 2023. Disponível em: <https://www.jusbrasil.com.br/artigos/entidades-privadas-tem-o-dever-legal-de-prestar-informacoes-assim-como-orgaos-publicos/1914573057>. Acesso em: 16 jun. 2025.

BRASIL. Lei n.º 12.527, de 18 de novembro de 2011. Regulamenta o acesso a informações previsto no inciso XXXIII do art. 5º, no inciso II do § 3º do art. 37 e no § 2º do art. 216 da Constituição Federal. Diário Oficial da União, Brasília, 18 nov. 2011. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2011-2014/2011/lei/l12527.htm>. Acesso em: 16 jun. 2025.

BRASIL. Lei n.º 8.069, de 13 de julho de 1990. Estatuto da Criança e do Adolescente. Diário Oficial da União, Brasília, 16 jul. 1990. Disponível em: <https://www.planalto.gov.br/ccivil_03/leis/l8069.htm>. Acesso em: 16 jun. 2025.

BRASIL. Lei n.º 9.394, de 20 de dezembro de 1996. Estabelece as diretrizes e bases da educação nacional. Diário Oficial da União, Brasília, 20 dez. 1996. Disponível em: <https://www.planalto.gov.br/ccivil_03/leis/l9394.htm>. Acesso em: 16 jun. 2025.

BRASIL. Lei n.º 13.146, de 6 de julho de 2015. Institui a Lei Brasileira de Inclusão da Pessoa com Deficiência (Estatuto da Pessoa com Deficiência). Diário Oficial da União, Brasília, 7 jul. 2015. Disponível em: <https://www.planalto.gov.br/ccivil_03/leis/l9394.htm>. Acesso em: 16 jun. 2025.

CRB6. Curitiba é a capital brasileira com maior número de bibliotecas públicas. CRB6, s.d. Disponível em: <https://crb6.org.br/materias/curitiba-e-a-capital-brasileira-com-maior-numero-de-bibliotecas-publicas/#:~:text=S%C3%A3o%20184%20unidades%20incluindo%20as,popula%C3%A7%C3%A3o%20de%201.851.215%20habitantes>. Acesso em: 30 jun. 2025.

BRITTO, Felipe. Documentação de software: O que é, tipos e boas práticas. Sydle, 16 dez. 2024. Disponível em: <https://www.sydle.com/br/blog/documentacao-de-software-67607a278f7ac06b8fb6bbcc>. Acesso em: 04 jul. 2025.

SCIENCEDIRECT. Object Management Group. ScienceDirect, 2010. Disponível em: <https://www.sciencedirect.com/topics/computer-science/object-management-group>. Acesso em: 05 jul. 2025.

SANKHYA. BPMN: O que é? Qual o seu objetivo e como usar. Sankhya, 28 maio 2025. Disponível em: <https://www.sankhya.com.br/blog/bpmn/>. Acesso em: 06 jul. 2025.

ALMEIDA, Vinucius Nòbelie De. Elementos do BPMN: Entenda as atividades. Euax, 29 abr. 2016. Disponível em: <https://www.euax.com.br/2016/04/elementos-de-bpmn-entenda-as-atividades/?gad_source=1&gad_campaignid=22588119313&gbraid=0aaaaadywqorwx8aluk8tsarlms8y51uys&gclid=cj0kcqjw-nfdbhdyarisad-ilecihuaevphpuuoggeepokcnon-yuahqd9jveczeat1xqfory4b_qicaaow4ealw_wcb>. Acesso em: 06 jul. 2025.

TOTVS. BPMN: entenda o que é a modelagem de processo de negócios, como fazer e sua importância! TOTVS, 01 mar. 2023. Disponível em: <https://www.totvs.com/blog/gestao-industrial/bpmn/>. Acesso em: 06 jul. 2025.

MARIANO, Paulo. Diagrama: O Que É? Como Fazer? Explicação Detalhada Com Exemplos. Go Prático, s.d. Disponível em: <https://gopratico.com.br/blog/o-que-e-diagrama/>. Acesso em: 08 jul. 2025.

MARCHIORI, Lucas. Node.JS: o que é e como funciona? Blogbetrybe, 21 out. 2022. Disponível em: <https://blog.betrybe.com/nodejs/?utm_term=&utm_campaign=Performance+Max+Cursos&utm_source=adwords&utm_medium=ppc&hsa_acc=1466424558&hsa_cam=21861374146&hsa_grp=&hsa_ad=&hsa_src=x&hsa_tgt=&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gad_source=1&gad_campaignid=21850942791&gbraid=0AAAAACnih53fSQo1fOi1f2ywV8PeOXBDG&gclid=Cj0KCQjw-NfDBhDyARIsAD-ILeBsj_YkkU6pFHy-EIN0Jq19aR7R8OV0X98hVtxcjsAqzw9ppVtbEDAaAuRUEALw_wcB>. Acesso em: 08 jul. 2025.