FACULDADE DE TECNOLOGIA DE PRESIDENTE PRUDENTE
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DIOGO DE LIMA SCARMAGNANI

SISTEMA DE AGENDAMENTO DE BARBEARIA

Presidente Prudente – SP

2024

FACULDADE DE TECNOLOGIA DE PRESIDENTE PRUDENTE
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DIOGO DE LIMA SCARMAGNANI

SISTEMA DE AGENDAMENTO DE BARBEARIA

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Presidente Prudente, como requisito parcial para obtenção do diploma de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Me. Marcelo Buscioli Tenório

Presidente Prudente – SP

2024

DIOGO DE LIMA SCARMAGNANI

SISTEMA DE AGENDAMENTO DE BARBEARIA

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Presidente Prudente, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Aprovado em: 06 de dezembro de 2024.

BANCA EXAMINADORA

_____
Orientador: Prof. Me. Marcelo Buscioli Tenório
Faculdade de Tecnologia de Presidente Prudente
Presidente Prudente – SP

_____
Profª. Mª. Vanessa dos Anjos Borges
Faculdade de Tecnologia de Presidente Prudente
Presidente Prudente – SP

_____
Prof. Me. Dione Jonathan Ferrari
Faculdade de Tecnologia de Presidente Prudente
Presidente Prudente – SP

RESUMO

SCARMAGNANI, Diogo. Sistema de Agendamento de Barbearia. Orientador: Marcelo Buscioli Tenório. Ano. 2024. 45 f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Faculdade de Tecnologia de Presidente Prudente, Presidente Prudente, SP, 2024.

Este trabalho apresenta o desenvolvimento de um sistema de agendamento voltado para barbearias, com o objetivo de otimizar o gerenciamento de horários e melhorar a experiência dos usuários, incluindo clientes e barbeiros. O sistema foi implementado utilizando a arquitetura de software do tipo *Três Camadas*, seguindo princípios do *Clean Architecture*, e foi dividido em *backend* e *frontend*. O *backend* foi desenvolvido em *ASP.NET Core*, hospedado no *Railway*, utilizando *PostgreSQL* como banco de dados relacional, enquanto o *frontend* foi implementado com *React* e *Vite*, hospedado no *Netlify*. O sistema permite que clientes realizem login, pesquisem serviços e horários disponíveis, e agendem serviços de forma simples e intuitiva. Para os barbeiros, oferece funcionalidades para login, cadastro de serviços e consulta de agendamentos, garantindo um gerenciamento eficiente. O diagrama de casos de uso e a modelagem de classes auxiliaram no planejamento e construção da aplicação, reforçando o alinhamento entre requisitos e funcionalidades. Durante a implantação, foram utilizados serviços de hospedagem modernos, garantindo acessibilidade e escalabilidade. O desenvolvimento seguiu boas práticas de organização, com divisão clara das responsabilidades: a camada Domain contém as entidades; a camada *Repository* gerencia a interação com o banco de dados; a camada *Service* realiza a abstração e manipulação de dados; e a camada *Endpoint* fornece os *endpoints* para comunicação externa. Conclui-se que o sistema proporciona uma solução prática para o gerenciamento de agendamentos em barbearias, beneficiando tanto os clientes quanto os barbeiros, com destaque para sua simplicidade de implantação e utilização.

Palavras-chave: agendamento; barbearia; sistema *web*; *ASP.NET Core*; *React*.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

Nos últimos anos, o setor de serviços tem buscado inovações tecnológicas que otimizem o atendimento ao cliente e promovam uma gestão mais eficiente dos negócios. Um dos segmentos beneficiados por essa evolução é o de barbearias, onde sistemas de agendamento se tornam essenciais para organizar horários, evitar conflitos na agenda e melhorar a experiência do cliente.

Este trabalho apresenta o desenvolvimento de um sistema de agendamento para barbearias, que visa facilitar tanto o gerenciamento de serviços por parte dos barbeiros quanto o processo de marcação de horários pelos clientes. O sistema foi concebido para operar de forma prática e intuitiva, garantindo uma experiência satisfatória aos usuários. Nele, os clientes podem se cadastrar, consultar serviços disponíveis, e agendar horários com os barbeiros. Já os barbeiros podem gerenciar suas informações, como horários de trabalho e serviços oferecidos.

A pesquisa e o desenvolvimento desse sistema foram motivados pela crescente demanda por soluções que combinem eficiência operacional e facilidade de uso, especialmente para pequenos negócios. A implementação dessa ferramenta objetiva não apenas resolver problemas práticos do dia a dia de uma barbearia, mas também contribuir para a modernização de processos nesse setor.

A estrutura desta monografia está organizada em capítulos que detalham a abordagem adotada. O Capítulo 1 introduz o tema, delimita o escopo e apresenta os objetivos e a justificativa do trabalho. O Capítulo 2 descreve o produto desenvolvido, destacando suas funções, público-alvo e limitações. O Capítulo 3 apresenta os requisitos específicos do sistema, incluindo diagramas e especificações técnicas. Por fim, o Capítulo 4 aborda o projeto do software, com modelagens, diagramas e layouts que ilustram as escolhas arquiteturais e funcionais.

1.1 OBJETIVO

Este documento tem como objetivo principal apresentar uma Especificação de Requisitos de Software (ERS) para o sistema de agendamento de barbearias, com base nas diretrizes da norma IEEE 830/1998. Ele detalha os requisitos funcionais e

não funcionais do sistema, de forma a alinhar as expectativas entre o cliente, os analistas e os desenvolvedores, além de garantir uma compreensão clara do escopo e da utilização do software.

Adicionalmente, este trabalho buscou proporcionar ao autor um aprofundamento técnico e prático no desenvolvimento de sistemas web, contribuindo para o aprimoramento das habilidades em programação com *C#*, *ASP.NET Core*, *React* e *TypeScript*, bem como no entendimento e transformação de requisitos em soluções tecnológicas. Este aprendizado consolida não apenas competências técnicas, mas também a capacidade de análise crítica e resolução de problemas, fundamentais para a atuação profissional na área de desenvolvimento de software.

1.2 ESCOPO

O produto de software a ser desenvolvido, denominado **iCorte**, tem como objetivo principal facilitar o gerenciamento de agendamentos e serviços em barbearias. O sistema busca automatizar tarefas operacionais, como a organização da agenda, o registro de serviços e a comunicação com os clientes, permitindo uma gestão mais eficiente e transparente.

Este software foi projetado para eliminar as dificuldades enfrentadas na organização manual de horários, reduzir conflitos de agendamento e melhorar a experiência tanto dos clientes quanto dos profissionais. Além disso, o sistema oferecerá funcionalidades adicionais, como o controle financeiro e a visualização de dados estatísticos relevantes para a barbearia.

O que o sistema fará?

- Permitir que clientes agendem serviços online, consultando horários disponíveis.
- Registrar e organizar serviços oferecidos pela barbearia.
- Gerenciar perfis de usuários (clientes, barbeiros e administradores) com permissões específicas.
- Oferecer relatórios e métricas relacionadas aos agendamentos e faturamento.
- Gerenciar pagamentos relacionados aos serviços agendados.

O que o sistema não fará?

- Não incluirá funcionalidades para marketing avançado, como campanhas de e-mail ou publicidade integrada.
- Não suportará, inicialmente, a integração com sistemas externos de pagamento automático, limitando-se ao registro manual.

Objetivos específicos

- Automatizar 100% do processo de agendamento de serviços.
- Reduzir o tempo gasto no gerenciamento de agendas em até 50%.
- Melhorar a experiência do cliente, oferecendo maior flexibilidade e conveniência.
- Fornecer uma interface amigável e acessível tanto no ambiente web quanto em dispositivos móveis.
- Aumentar a capacidade de gerenciamento da barbearia por meio de relatórios intuitivos e detalhados.

1.3 DEFINIÇÕES, SIGLAS E ABREVIAÇÕES

- **ERS**: Especificação de Requisitos de *Software*;
- **ORM**: *Object-Relational Mapping* (Mapeamento Objeto-Relacional);
- **DTO**: *Data Transfer Object* (Objeto de Transferência de Dados);
- **API**: *Application Programming Interface* (Interface de Programação de Aplicações);
- **ASP.NET Core**: *Framework* de desenvolvimento para aplicações web da Microsoft;
- **Vite**: Ferramenta de build e desenvolvimento rápido para *frontends* modernos;
- **CRUD**: *Create*, *Read*, *Update*, *Delete* (Criar, Ler, Atualizar, Deletar);
- **Git**: Sistema de controle de versão distribuído;
- **GitHub**: Plataforma de hospedagem de código baseada em *Git*;
- **Deployment:** Processo de colocar uma aplicação em um ambiente onde ela pode ser usada por usuários finais;

- **Build:** Processo de transformar código-fonte em um executável ou pacote pronto para *deployment*;
- **Railway**: Plataforma para hospedagem de aplicações web e bancos de dados;
- **Netlify**: Plataforma para hospedagem e *deploy* de aplicações *frontend*;
- **EF Core**: *Entity Framework Core* (*Framework* de ORM para .NET);
- **REST**: *Representational State Transfer* (Transferência Representacional de Estado);
- **TCC**: Trabalho de Conclusão de Curso;
- **IDE**: *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado);
- **UML**: *Unified Modeling Language* (Linguagem de Modelagem Unificada);
- **SQL**: *Structured Query Language* (Linguagem de Consulta Estruturada).

1.4 VISÃO GERAL

Esta *ERS* está organizada em capítulos, cada um contendo informações detalhadas sobre os aspectos fundamentais do sistema em desenvolvimento. A estrutura foi elaborada para garantir clareza e facilidade de entendimento por todos os stakeholders envolvidos no projeto.

A seguir, apresentamos um resumo de como o documento está estruturado:

- **Capítulo 2:** Este capítulo fornece uma visão geral do sistema, apresentando uma perspectiva do produto, suas principais funções, o perfil dos usuários que utilizarão o software, características gerais do desenvolvimento e os requisitos que foram adiados para versões futuras.
- **Capítulo 3:** Neste capítulo, são detalhados os requisitos do sistema, incluindo os requisitos de interface externa, o diagrama de casos de uso, as especificações de cada caso de uso fundamental e os diagramas de atividades associados.
- **Capítulo 4:** Este capítulo descreve o projeto do software, incluindo os diagramas de interação (sequência), o diagrama de classes, a modelagem da base

de dados, o diagrama de pacotes da arquitetura lógica e outros layouts relevantes para a implementação do sistema.

Cada seção foi planejada para oferecer uma visão clara e abrangente sobre os aspectos técnicos e funcionais do software, assegurando que as informações sejam acessíveis e úteis tanto para desenvolvedores quanto para gestores e usuários finais.

2. DESCRIÇÃO GERAL DO PRODUTO

Este capítulo fornece uma visão geral do sistema de agendamento de barbearias, abordando suas principais funcionalidades, público-alvo e aspectos gerais do seu desenvolvimento.

O sistema foi concebido para facilitar o gerenciamento de agendamentos e otimizar a interação entre clientes e barbeiros. Ele permite que os usuários se cadastrem como clientes ou barbeiros, possibilitando que cada perfil desempenhe funções específicas:

- **Clientes**: podem navegar pelos serviços oferecidos, consultar a disponibilidade dos barbeiros e agendar horários de forma prática.
- **Barbeiros**: têm a capacidade de cadastrar seus serviços, configurar seus horários de trabalho e gerenciar os agendamentos realizados pelos clientes.

Além disso, o sistema conta com um mecanismo de avaliação para que os clientes possam atribuir notas e comentários aos serviços recebidos, promovendo uma dinâmica de reputação. Embora uma funcionalidade de chat tenha sido planejada para facilitar a comunicação direta entre clientes e barbeiros, sua implementação foi adiada para versões futuras.

O desenvolvimento deste sistema foi fundamentado em tecnologias modernas e robustas, incluindo *ASP.NET Core* no *backend*, *React* com *TypeScript* no *frontend*, e o banco de dados *PostgreSQL*. A infraestrutura foi hospedada utilizando *Railway* (*backend* e banco de dados) e *Netlify* (*frontend*), garantindo confiabilidade e escalabilidade para o ambiente de produção.

Essa descrição geral fornece o contexto necessário para a análise e detalhamento dos requisitos específicos no próximo capítulo, ressaltando o potencial do sistema como uma solução prática e escalável para barbearias de diversos portes.

2.1 FUNÇÕES DO SISTEMA

Funções Básicas

F_B01: Cadastrar Conta: Permite ao usuário criar uma conta no sistema como cliente ou barbeiro.

Dados de entrada: Nome, sobrenome, telefone, gênero, email, senha.

Regra de negócio: Senha que atenda aos requisitos mínimos de segurança.

Dados de saída: Conta criada com sucesso, acesso autorizado ao sistema.

F_B02: Cadastrar Barbearia: Permite o registro de informações de uma barbearia vinculada ao barbeiro.

Dados de entrada: Nome, descrição, telefone comercial, email comercial, endereço.

Regra de negócio: Descrição é opcional; telefone e email comerciais podem ser os mesmos do perfil de cliente.

Dados de saída: Barbearia cadastrada com sucesso, acesso autorizado à parte exclusiva a barbeiros.

Funções Fundamentais

F_F01: Login: Permite que o usuário acesse o sistema utilizando suas credenciais.

Dados de entrada: Email e senha.

Regra de negócio: *F_B01* precisa já ter sido executada.

Dados de saída: Login efetuado e cookie com token de autenticação recebido e persistido do lado do cliente.

F_F02: Cadastrar Serviços: Habilita o barbeiro a incluir e gerenciar os serviços que oferece.

Dados de entrada: Nome do serviço, preço, tempo de execução.

Regra de negócio: *F_B02* precisa já ter sido executada.

Dados de saída: Serviço criado com sucesso.

F_F03: Agendar Serviço: Permite ao cliente selecionar um serviço, escolher um horário disponível e realizar o agendamento.

Dados de entrada: Serviço que deseja agendar, data, horário de início, forma de pagamento.

Regra de negócio: *F_S01* precisa ter sido checado.

Dados de saída: Serviço agendado com sucesso.

Funções de Saída

F_S01: Pesquisar Serviços e Horários Disponíveis: Permite ao cliente visualizar os serviços disponíveis e os horários associados.

Dados de entrada: Data e serviço(s) desejado(s).

Regra de negócio: Serviço precisa existir.

Dados de saída: Exibe os intervalos de horários disponíveis para aquele(s) serviço(s) escolhido(s).

F_S02: Meus Agendamentos: Exibe uma listagem dos agendamentos realizados ou a realizar, funcionando como um relatório para o barbeiro.

Dados de entrada: Data.

Regra de negócio: *F_B02* precisa ter sido executada.

Dados de saída: Lista com agendamentos existentes.

2.2 CARACTERÍSTICAS DO USUÁRIO

O sistema foi desenvolvido para atender dois perfis principais de usuários: Clientes e Barbeiros.

- **Clientes**: Os clientes representam o público mais amplo do sistema. Eles são indivíduos alfabetizados que possuem acesso a dispositivos conectados à internet, como smartphones, tablets ou computadores. Não é exigido conhecimento técnico específico em informática, já que o sistema foi projetado para ser intuitivo e acessível, permitindo que qualquer pessoa consiga navegar e realizar agendamentos com facilidade.
- **Barbeiros**: Os barbeiros, além de possuírem características similares às dos clientes, como acesso a dispositivos conectados à internet e alfabetização, necessitam de habilidades específicas relacionadas ao exercício da profissão. Eles também devem ser capazes de cadastrar seus serviços e gerenciar horários no sistema, tarefas que demandam um entendimento básico de informática e operação de aplicativos.

O design do sistema prioriza a usabilidade e a simplicidade, reduzindo a necessidade de treinamento técnico específico para ambos os perfis.

2.3 LIMITES, DEPENDÊNCIAS E SUPOSIÇÕES

Limites

1. O sistema foi projetado para ser 100% web, sendo assim, depende de conexão estável com a internet tanto para clientes quanto para barbeiros. A ausência de conectividade pode limitar a experiência de uso e o acesso às funcionalidades.

2. O desempenho do sistema está diretamente ligado à infraestrutura de hospedagem (*Railway* para *backend* e banco de dados, e *Netlify* para o *frontend*). Qualquer indisponibilidade desses serviços pode impactar no funcionamento do sistema.

3. Não há suporte nativo para funcionamento offline ou integração com dispositivos específicos, como impressoras de recibo ou leitores de *QR Code*.

Dependências

1. O sistema utiliza tecnologias modernas como *ASP.NET Core***,** *React* com *TypeScript* e *PostgreSQL*, o que requer que qualquer futura manutenção ou atualização seja feita por profissionais com conhecimento nessas tecnologias.

2. A hospedagem na plataforma *Railway* e *Netlify* é uma dependência crítica. Caso o suporte a essas plataformas seja descontinuado, será necessário migrar o sistema para outros provedores compatíveis.

3. A experiência dos usuários está atrelada ao navegador utilizado. O sistema é compatível com os navegadores modernos mais populares (*Google Chrome*, *Mozilla Firefox*, *Microsoft Edge*), mas problemas de compatibilidade podem ocorrer em navegadores obsoletos.

Suposições

1. Supõe-se que os usuários do sistema (clientes e barbeiros) tenham acesso a dispositivos conectados à internet, como smartphones, tablets ou computadores, e que possuam níveis básicos de alfabetização digital.

2. É assumido que o proprietário do sistema (o próprio desenvolvedor) será responsável por monitorar, corrigir falhas e realizar upgrades na infraestrutura e nas tecnologias utilizadas.

3. A escalabilidade do sistema dependerá do aumento dos recursos alocados nas plataformas de hospedagem, conforme o número de usuários e a demanda cresçam.

4. Não foram implementadas funcionalidades avançadas de segurança, como autenticação multifator. Supõe-se que as práticas básicas de segurança, como a utilização de senhas fortes, serão seguidas pelos usuários.

Esse conjunto de limites, dependências e suposições define o contexto em que o sistema foi desenvolvido e deverá operar, garantindo que as expectativas estejam alinhadas às capacidades atuais do software.

2.4 REQUISITOS ADIADOS

Os seguintes requisitos foram identificados como funcionalidades importantes, mas que serão adiados para versões futuras do sistema:

F_F04: Avaliar Serviços – Permitir que os clientes avaliem os serviços recebidos, atribuindo uma nota e comentários, para ajudar outros usuários na escolha do barbeiro e para o barbeiro acompanhar sua reputação.

F_F05: Chat Entre Cliente e Barbeiro – Implementar um sistema de comunicação direta entre cliente e barbeiro, permitindo tirar dúvidas e ajustar detalhes antes do agendamento.

Esses requisitos foram adiados devido a restrições de tempo e ao escopo inicial do projeto. Sua implementação futura dependerá da análise de viabilidade técnica e da demanda por parte dos usuários do sistema.

3. REQUISITOS ESPECÍFICOS

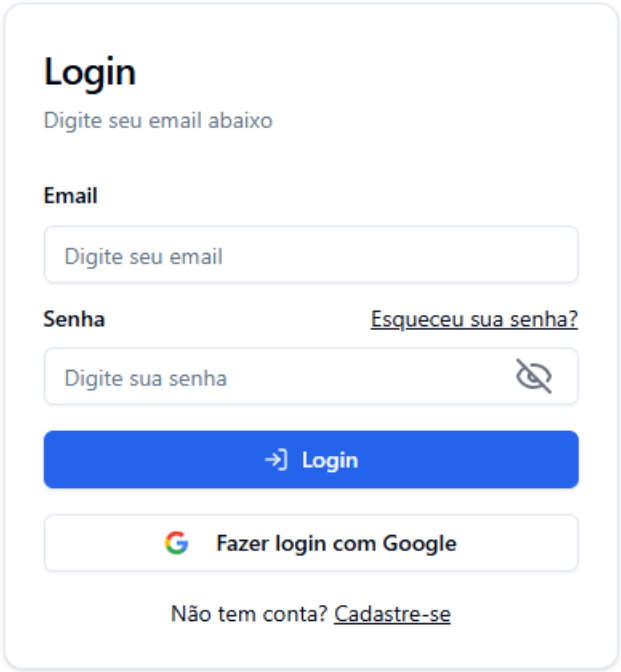
Este capítulo apresenta um detalhamento dos requisitos necessários para o funcionamento do sistema, garantindo que todas as funcionalidades e interações descritas estejam devidamente especificadas. Ele será estruturado em subseções que abordam os requisitos de interface externa, as especificações dos casos de uso e os diagramas associados, fornecendo uma visão clara e organizada das necessidades do sistema.

Cada subtítulo a seguir contém informações detalhadas que servirão como base para o desenvolvimento, implementação e validação do sistema.

3.1 REQUISITOS DE INTERFACE EXTERNA

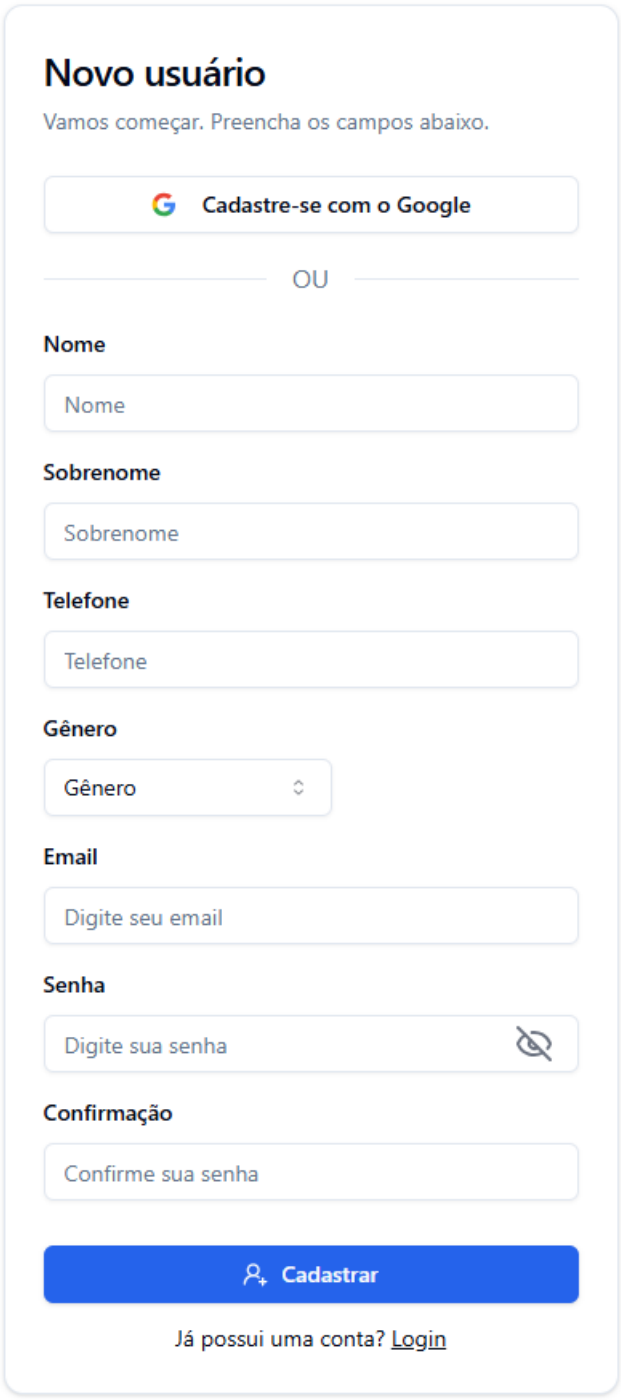
3.1.1 Interfaces do Usuário dos Casos de Uso

Figura 1 - Tela de Login



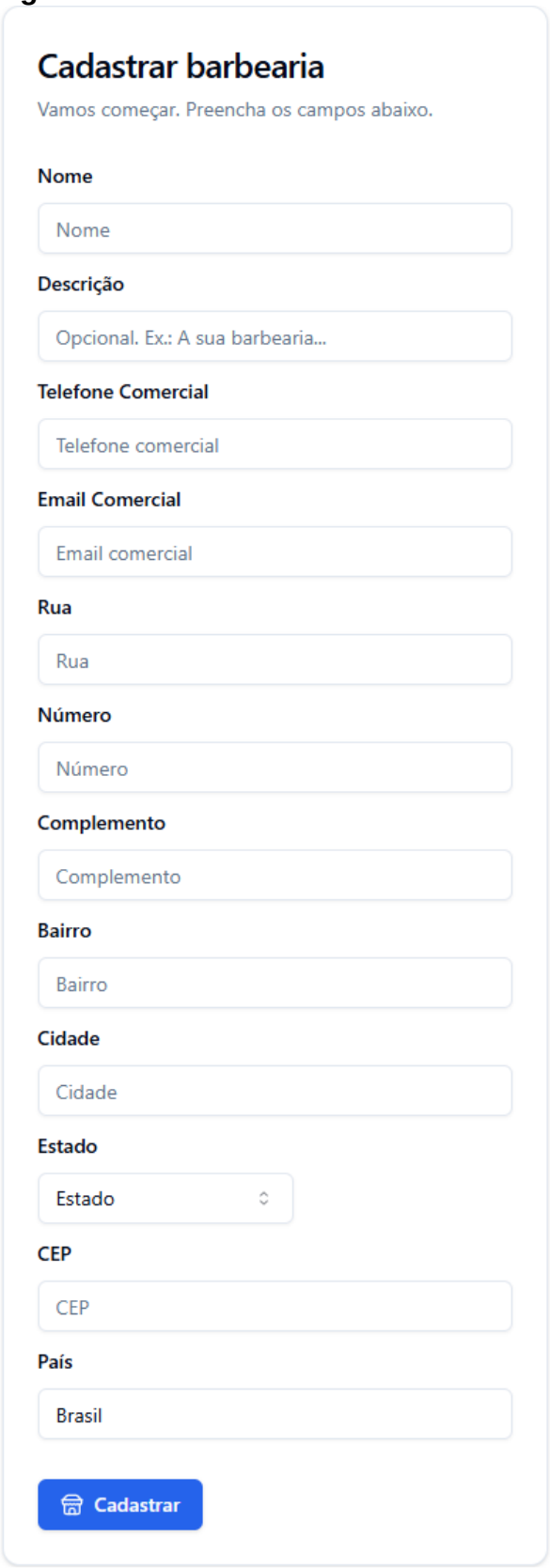
Fonte: Elaborado pelo próprio autor.

Figura 2 - Tela de Cadastro do Usuário



Fonte: Elaborado pelo próprio autor.

Figura 3 - Tela de Cadastro do Barbeiro



Fonte: Elaborado pelo próprio autor.

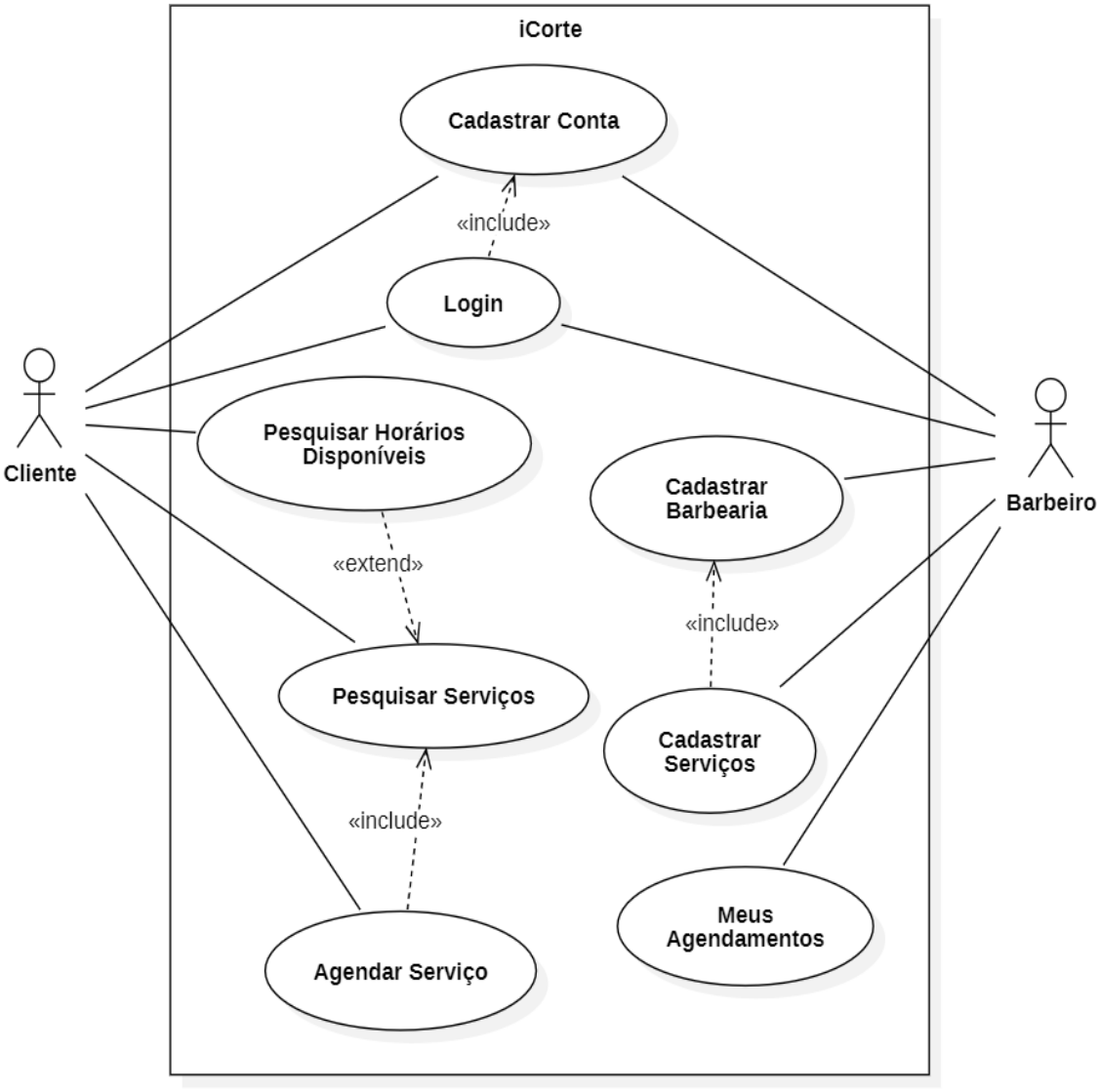
3.1.2 Interfaces de Software

1. **Banco de Dados:**
    - **PostgreSQL:** O sistema utiliza o banco de dados PostgreSQL, que foi acessado diretamente via terminal para gerenciamento de tabelas e dados.
    - **Entity Framework ORM:** Através da aplicação, toda comunicação entre a camada de *Repository* e a camada de *Service* é feita através do *ORM* nativo do *.NET* chamado *Entity Framework*.
2. **Sistema Operacional:**
    - **Windows 11:** Ambiente principal de desenvolvimento.
    - **Arch Linux (WSL):** Utilizado como subsistema para gerenciar ferramentas de desenvolvimento e banco de dados.
3. **IDE e Ferramentas de Desenvolvimento:**
    - **Visual Studio Code:** IDE utilizada para codificação e integração do sistema.
    - **GitHub:** Plataforma de versionamento de código e colaboração, onde todo o código-fonte foi armazenado e gerenciado.
4. **Hospedagem e Serviços Web:**
    - **Railway:** Serviço de hospedagem para o backend do sistema.
    - **Netlify:** Utilizado para hospedagem do frontend.

Esses softwares foram integrados para garantir um ambiente de desenvolvimento ágil e eficaz, além de fornecer suporte robusto para a execução do sistema.

3.2 DIAGRAMA DE CASOS DE USO

Figura 4 – Diagrama de Casos de Uso



Fonte**:** Elaborado pelo próprio autor.

3.3 ESPECIFICAÇÕES DOS CASOS DE USO

Caso de Uso 1: Login

Descrição: Permite que um usuário (cliente ou barbeiro) se autentique no sistema para acessar suas funcionalidades específicas.

Atores: Cliente, Barbeiro.

Pré-condição: O usuário já deve ter uma conta cadastrada no sistema.

Fluxo Principal:

1. O usuário acessa a página inicial do sistema.
2. Informa suas credenciais (e-mail e senha).
3. O sistema valida as credenciais.
4. Se as credenciais forem válidas, o sistema redireciona o usuário para seu painel específico (cliente ou barbeiro).

Fluxo Alternativo:

- Caso as credenciais sejam inválidas, o sistema exibe uma mensagem de erro e permite nova tentativa.

Pós-condição: O usuário está autenticado e pode acessar as funcionalidades específicas de seu perfil.

Caso de Uso 2: Cadastrar Serviços

Descrição: Permite que um barbeiro cadastre os serviços que ele oferece, incluindo o preço e o tempo de execução.

Ator: Barbeiro

Pré-condição: O barbeiro deve estar autenticado no sistema.

Fluxo Principal:

1. O barbeiro acessa o painel de controle.
2. Seleciona a opção “Cadastrar Serviços”.
3. Preenche os campos obrigatórios (nome do serviço, preço e duração).
4. Confirma a ação.
5. O sistema salva o serviço e exibe uma mensagem de sucesso.

Pós-condição: O serviço está disponível para visualização pelos clientes.

Caso de Uso 3: Agendar Serviço

Descrição: Permite que um cliente selecione um serviço e marque um horário com um barbeiro.

Ator: Cliente

Pré-condição: O cliente deve estar autenticado no sistema.

Fluxo Principal:

1. O cliente acessa a funcionalidade de agendamento.
2. Pesquisa por barbeiros e serviços disponíveis.
3. Seleciona um barbeiro, um serviço e um horário disponível.
4. Confirma o agendamento.
5. O sistema salva o agendamento e exibe a confirmação.

Fluxo Alternativo:

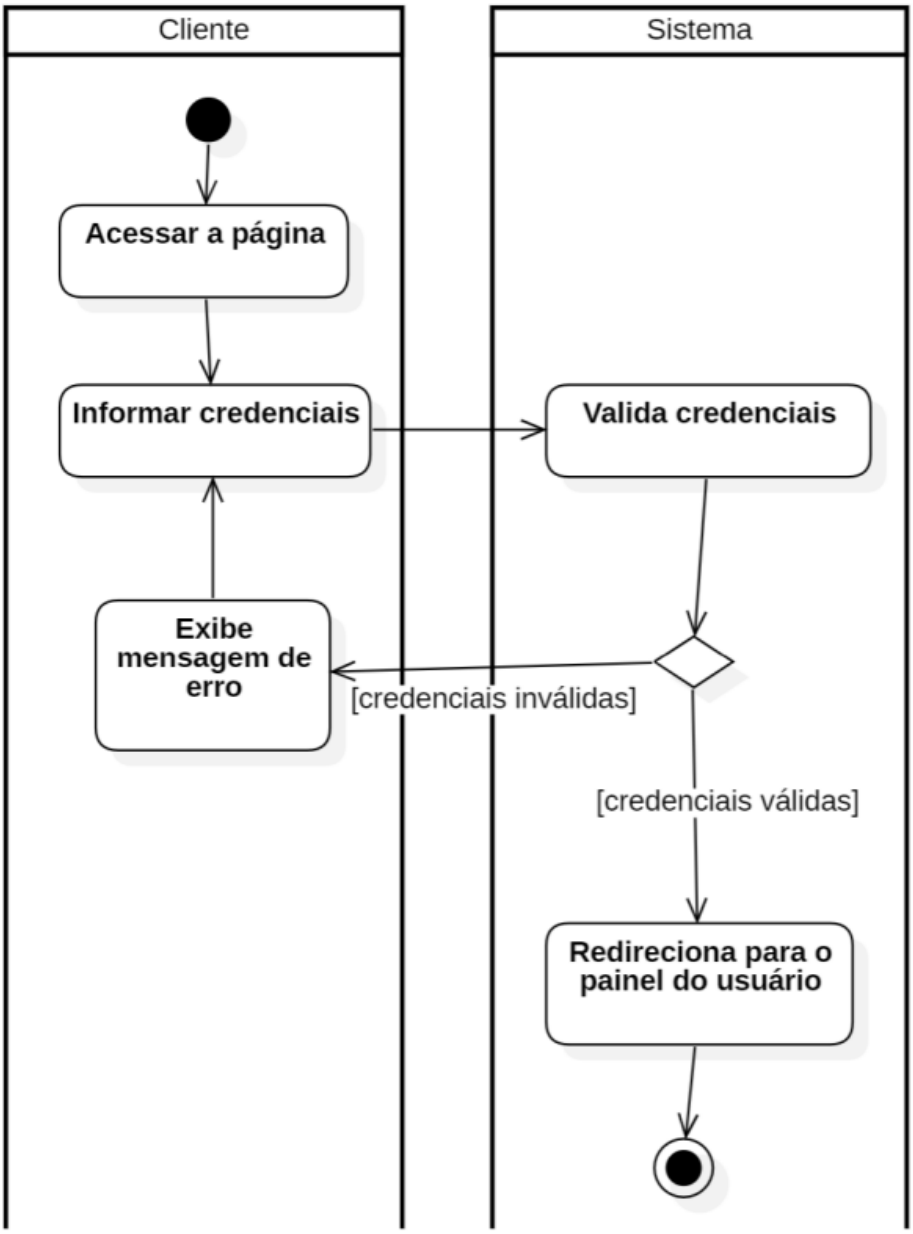
- Caso o horário escolhido não esteja mais disponível, o sistema solicita uma nova escolha.

Pós-condição: O agendamento está salvo e visível no painel do cliente e do barbeiro.

3.4 DIAGRAMAS DE ATIVIDADES DOS CASOS DE USO

Diagrama de Atividade - Login

Figura 5

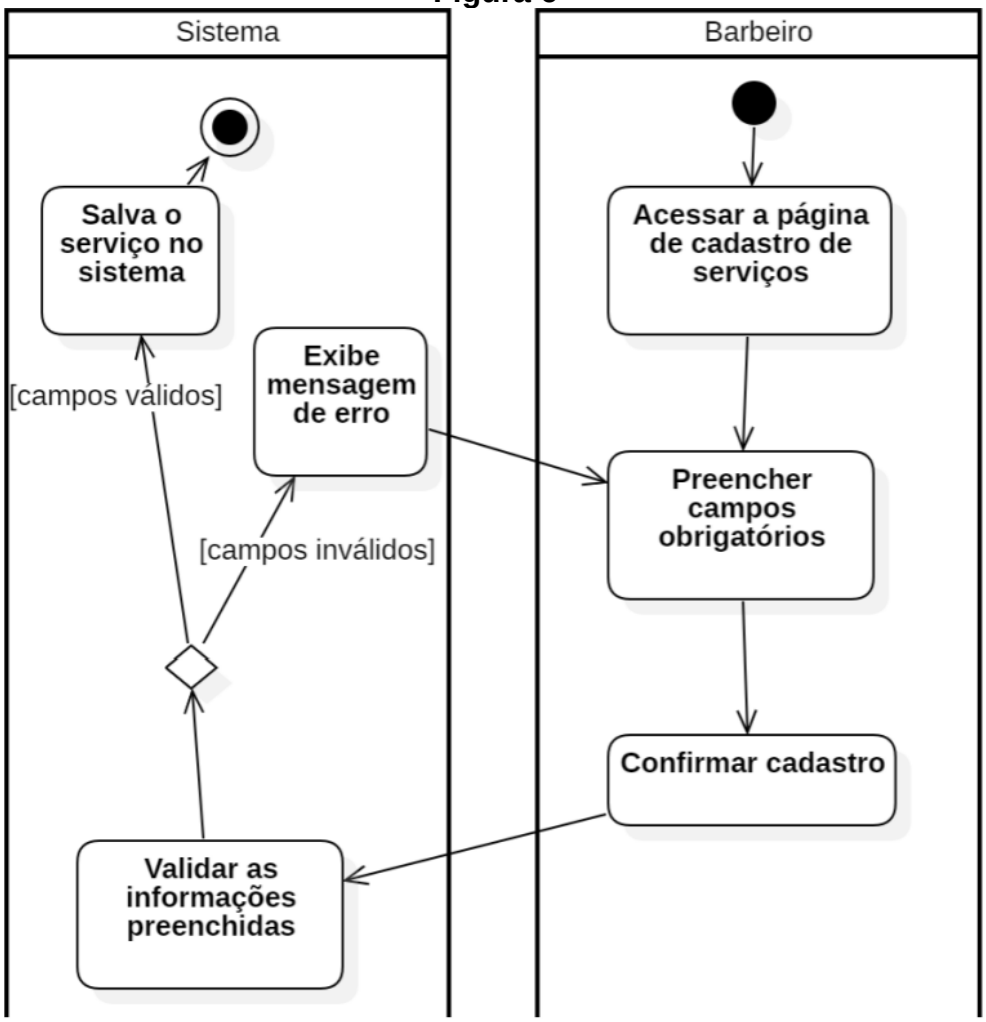


Fonte: Elaborado pelo próprio autor.

Descrição: Demonstra o fluxo de atividades que um usuário (cliente ou barbeiro) realiza para autenticar-se no sistema.

Diagrama de Atividade - Cadastrar Serviços

Figura 6

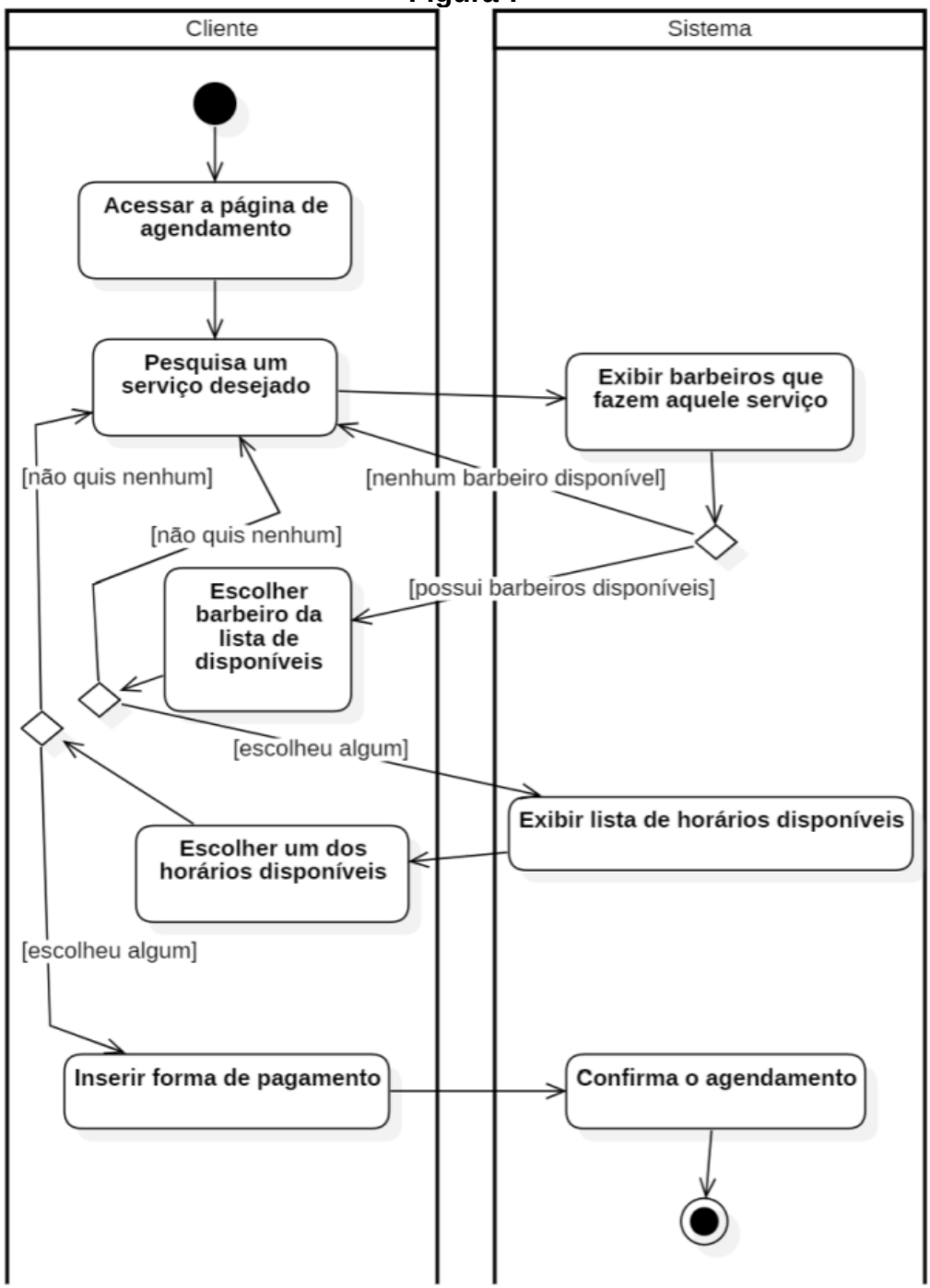


Fonte: Elaborado pelo próprio autor.

Descrição: Mostra o fluxo de atividades que um barbeiro executa para cadastrar os serviços os quais oferece no sistema.

Diagrama de Atividade - Agendar Serviço

Figura 7



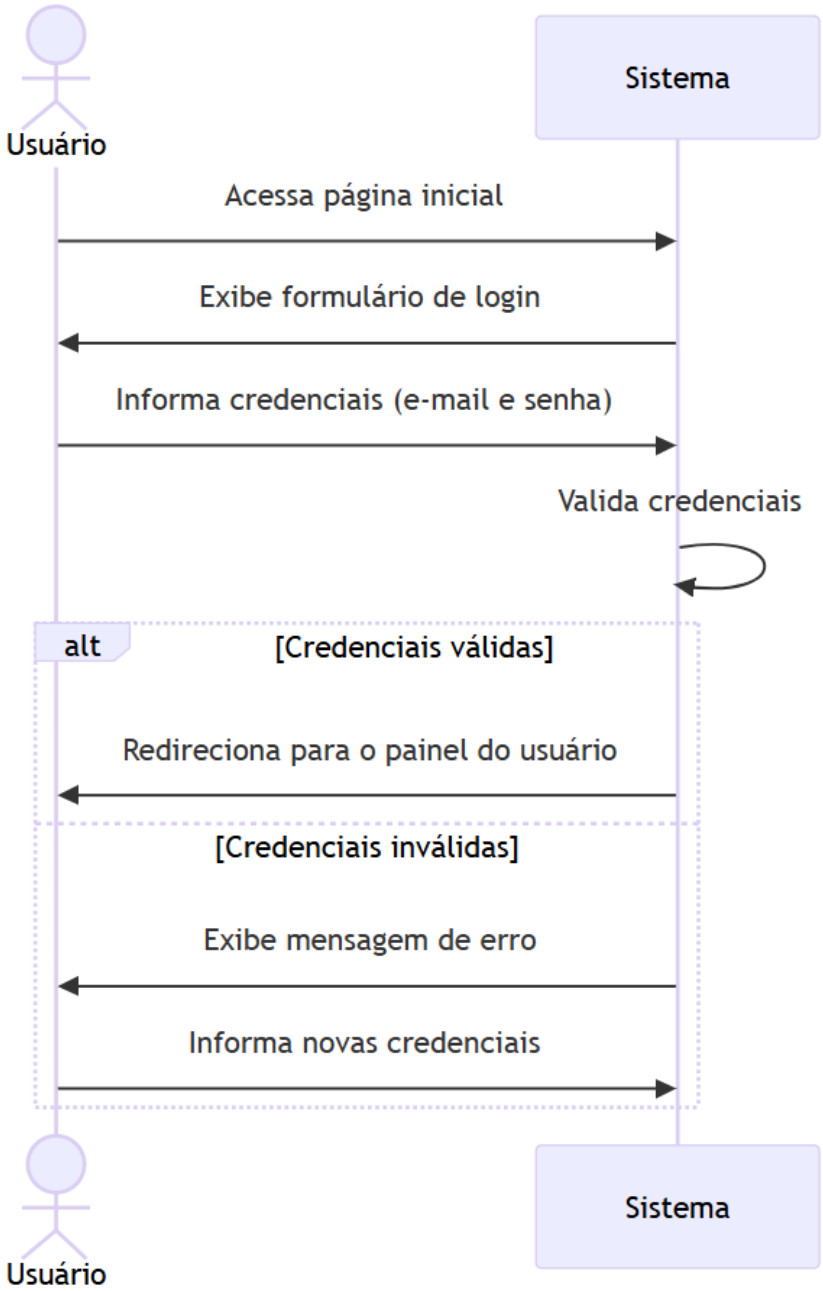
Fonte: Elaborado pelo próprio autor.

Descrição: Representa o fluxo de atividades que um cliente realiza para agendar um serviço com um barbeiro.

4. PROJETO DE SOFTWARE

4.1 DIAGRAMAS DE INTERAÇÃO

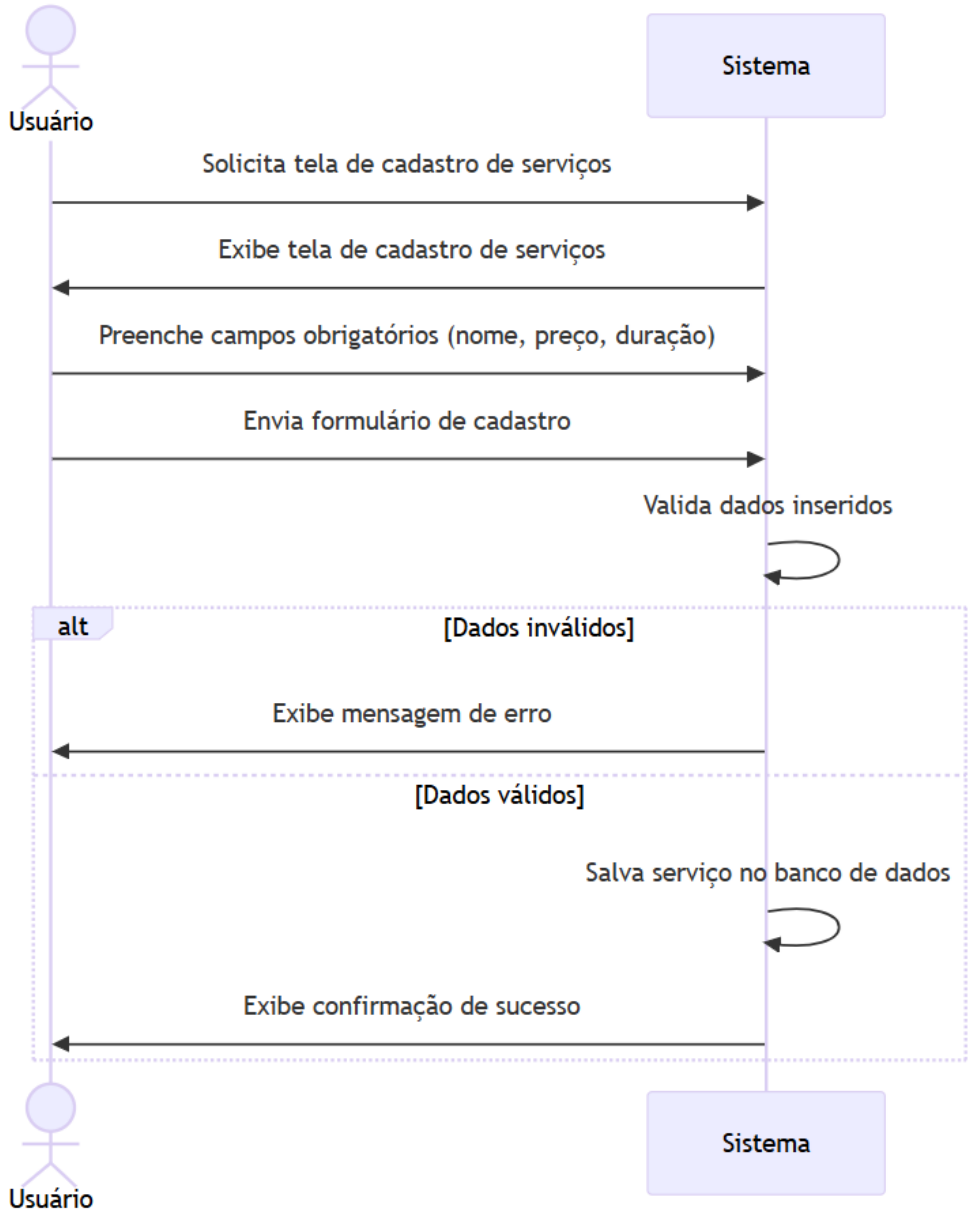
Figura 8 - Diagrama de Interação - Login



Fonte: Elaborado pelo próprio autor.

Descrição: Demonstra o fluxo de ações para que um usuário (cliente ou barbeiro) autentique-se no sistema.

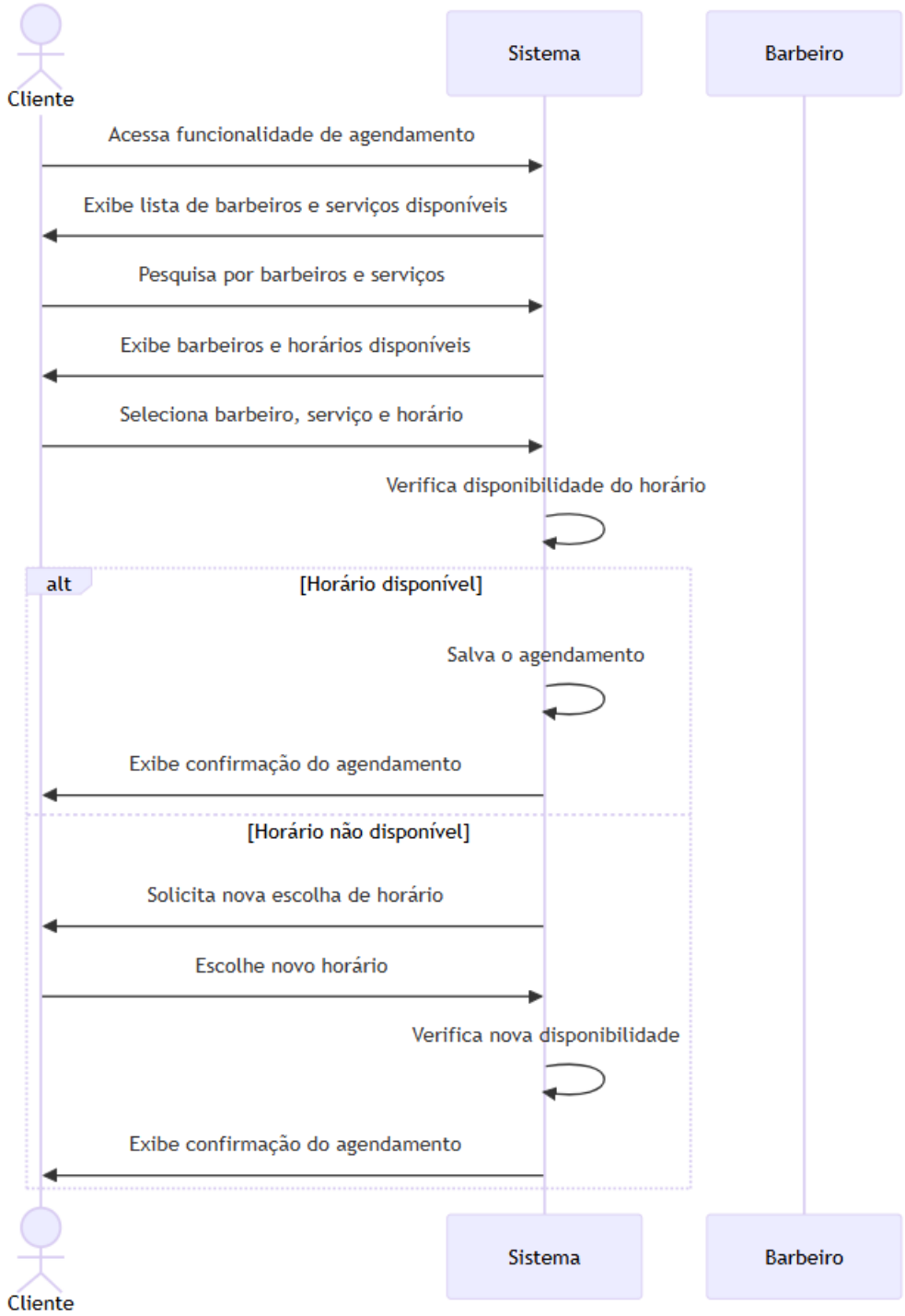
Figura 9 - Diagrama de Interação - Cadastrar Serviços



Fonte: Elaborado pelo próprio autor.

Descrição: Mostra o fluxo de ações que o barbeiro executa para cadastrar os serviços os quais oferece no sistema.

Figura 10 - Diagrama de Interação - Agendar Serviço

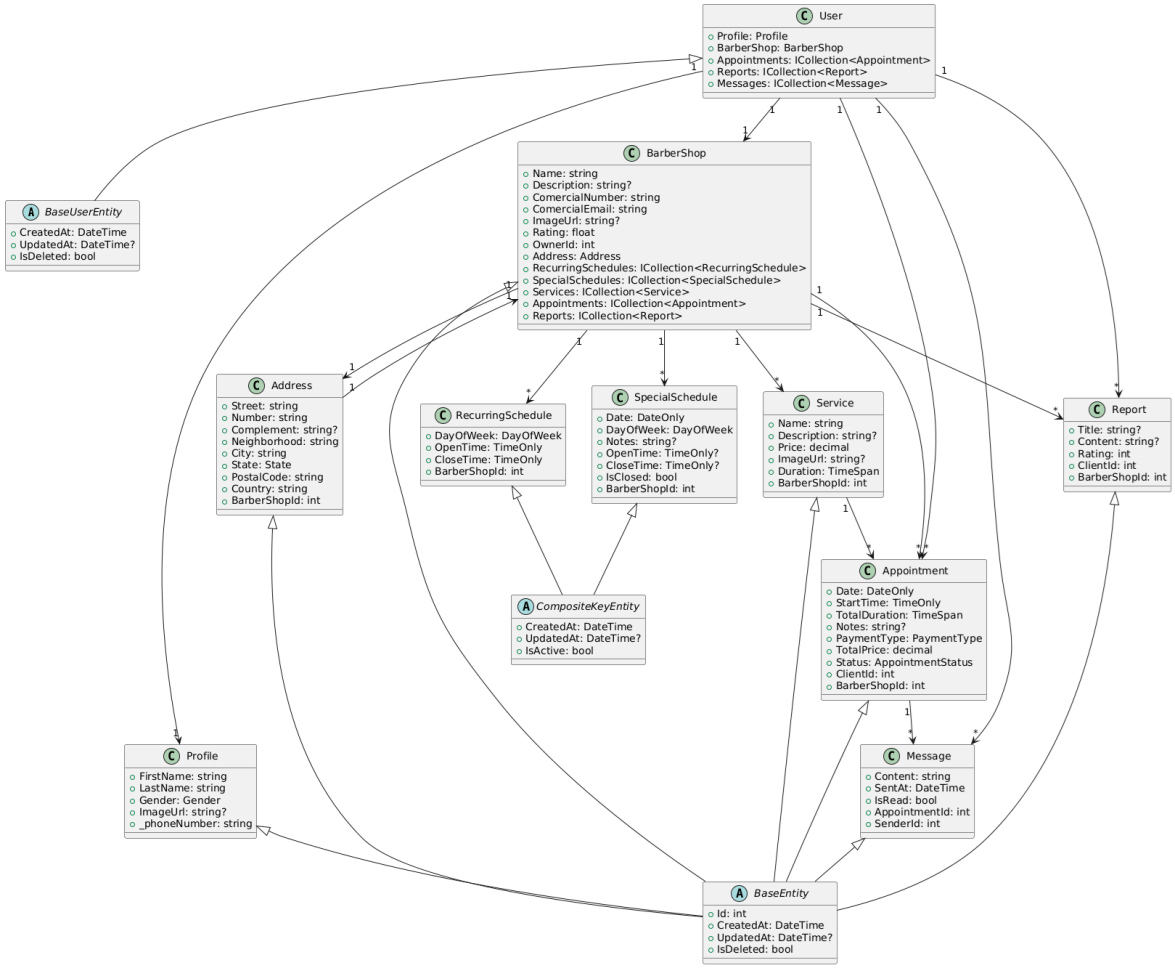


Fonte: Elaborado pelo próprio autor.

Descrição: Representa o fluxo de ações que um cliente realiza para agendar um serviço com um barbeiro.

4.2 DIAGRAMA DE CLASSES

Figura 11

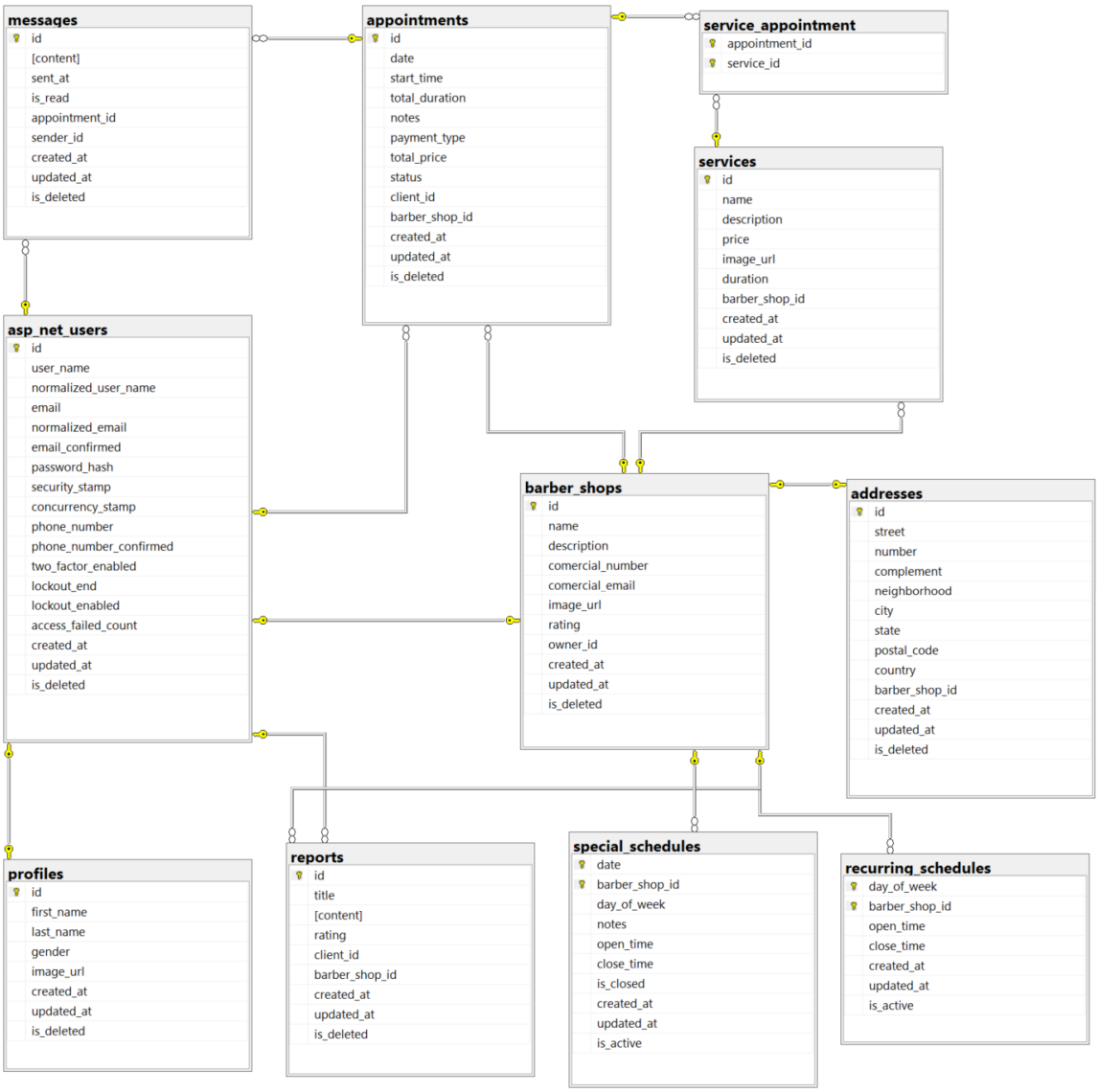


Fonte: Elaborado pelo próprio autor.

Descrição: Classes base para criação das entidades que são refletidas no banco de dados da aplicação. A grande maioria das classes herdam a classe abstrata *BaseEntity* onde encontramos propriedades padrões *Id*, *CreatedAt*, *UpdatedAt* e *IsDeleted*. As classes *RecurringSchedule* e *SpecialSchedule* referem-se aos horários disponíveis do barbeiro. As classes *Report* e *Message* são as classes que serão usadas para as avaliações e para o *chat* que são das funções dos requisitos adiados.

4.3 MODELAGEM DA BASE DE DADOS

Figura 12



Fonte: Elaborado pelo próprio autor.

4.4 DIAGRAMA DE PACOTES DA ARQUITETURA LÓGICA

O sistema foi projetado em **Três Camadas** para seguir uma abordagem modular e escalável, adotando princípios inspirados na *Clean Architecture* no *backend*. Essa divisão em camadas facilita a separação de responsabilidades, garantindo um desenvolvimento mais organizado e sustentável. Abaixo estão descritas as principais camadas e seus papéis na arquitetura:

***Domain*:** Esta camada contém as entidades do sistema, que são representações das tabelas no banco de dados. As entidades encapsulam a lógica central de negócio, como regras e validações fundamentais, e são utilizadas em toda a aplicação como uma base para os dados persistidos.

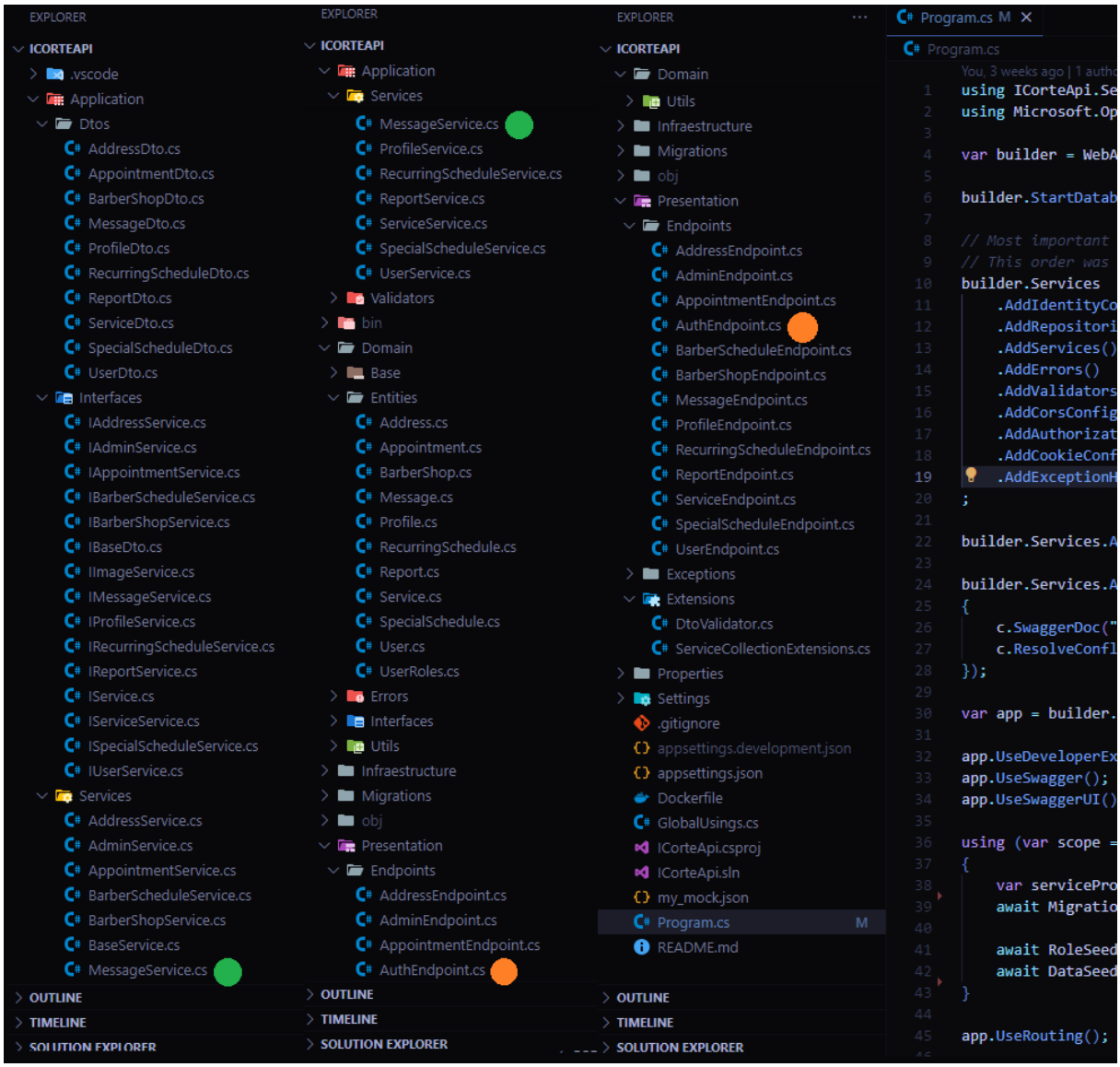
***Repository*:** Nesta camada, é realizada a comunicação com o banco de dados por meio do *EF Core*. Ela é responsável pelas operações de leitura, gravação, atualização e exclusão de dados, abstraindo a complexidade de interação com o banco e fornecendo métodos que podem ser utilizados pelas camadas superiores.

***Service*:** A camada de serviços serve como um intermediário entre o *Repository* e o *Endpoint*. Ela é responsável por processar as regras de negócio específicas, filtrar informações desnecessárias (como, por exemplo, *IsDeleted* e *UpdatedAt*, que são colunas de metadados), e transformar dados em *DTOs* para serem enviados ao cliente. Além disso, os serviços gerenciam as operações que exigem múltiplos acessos ao banco de dados, coordenando as chamadas ao repositório.

***Endpoints*:** Esta camada representa a *Presentation* e atua como o ponto de interação entre o mundo externo (no caso do sistema, o *frontend*, mas também poderia ser um serviço específico de teste de *API*, como o *Swagger* ou o *Postman*) e o *backend*. Responsável por expor os *endpoints* da *API*, ela recebe requisições, faz as devidas validações e as encaminha para os serviços correspondentes. Também trata as respostas, garantindo que sejam adequadas para serem entregues ao cliente, sejam mensagens de erro ou dados processados.

Essa organização em pacotes visa garantir alta coesão e baixo acoplamento entre as camadas, promovendo um design limpo, fácil de manter e expansível para futuras implementações. O diagrama de pacotes reflete essa estrutura, destacando como cada camada se relaciona e contribui para a lógica geral do sistema.

Figura 13 – Estrutura da *Solution*

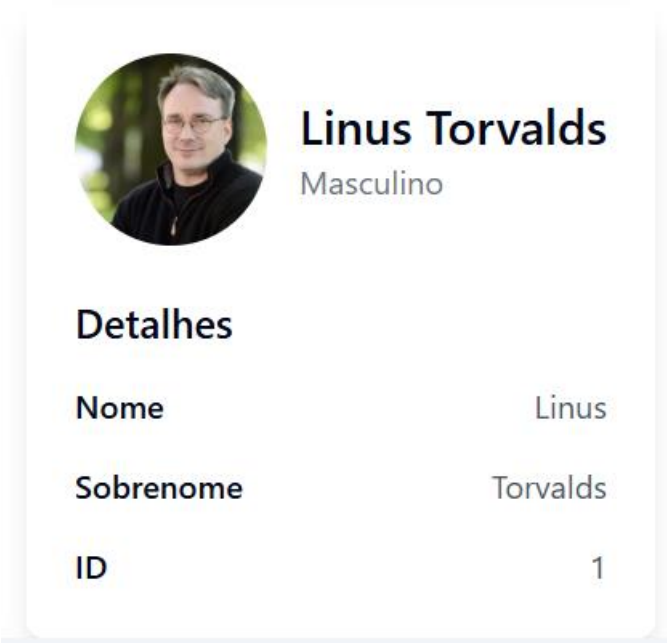


Fonte: Elaborado pelo próprio autor.

Descrição: Estrutura da *Solution*, destacando os *layers* de *Application*, *Domain* e *Presentation*. Ainda há o *layer* de *Infraestructure*, que a grosso modo espelha as *entities* para conversar com a cama externa (no caso dessa aplicação: apenas o banco de dados) através do *EF Core*.

4.5 OUTROS LAYOUTS DE TELAS

Figura 14 - Tela de Exibição do Perfil do Usuário



Fonte: Elaborado pelo próprio autor.

Figura 15 - Tela de Exibição do Perfil do Barbeiro

Barbearia do Torvalds

Aqui o corte é open source!

Informações de Contato

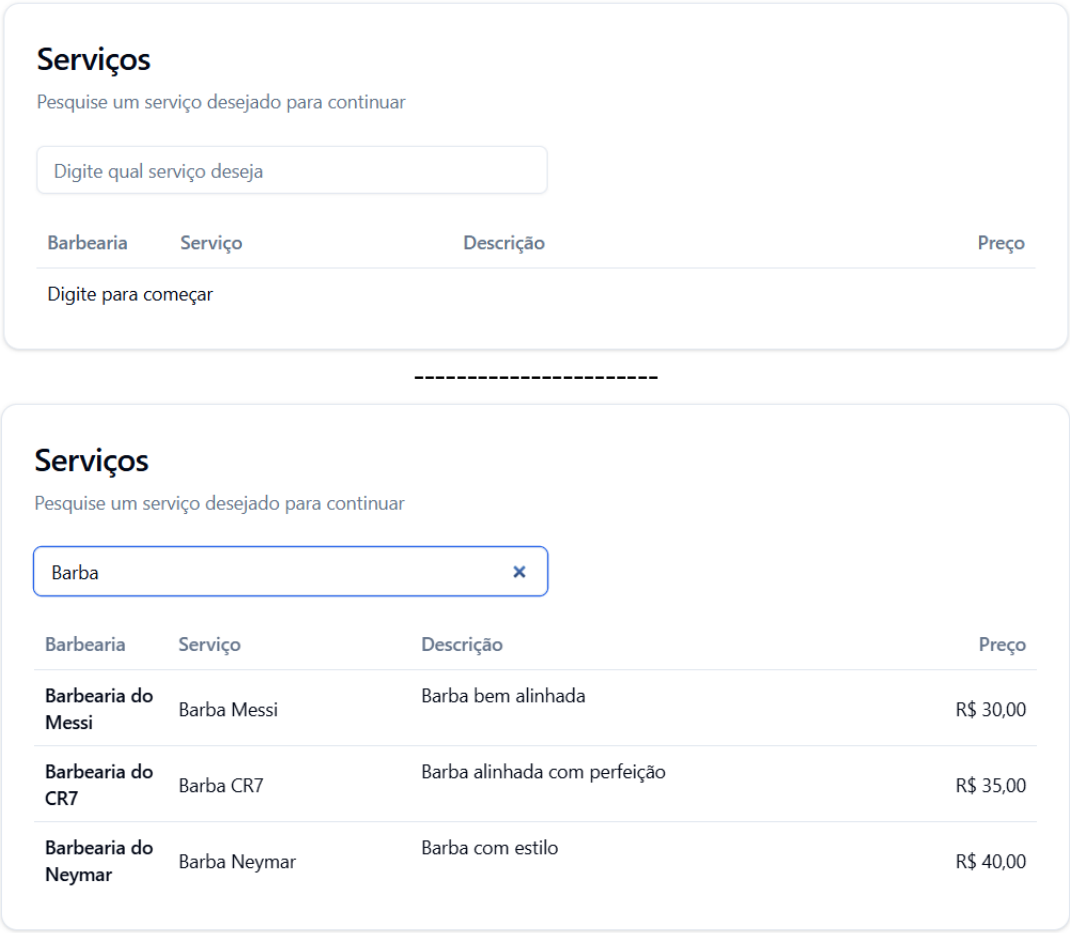
Número Comercial	(11) 96543-5265
E-mail Comercial	linus@torvalds.com

Endereço

Endereço	Rua Linux, 180
Bairro	Kernel Central
Cidade	São Paulo
Estado	SP
CEP	19180-000
País	Brasil

Fonte: Elaborado pelo próprio autor.

Figura 16 - Tela de Pesquisa de Serviços



Fonte: Elaborado pelo próprio autor.

APÊNDICE A – Procedimentos para Implantação do Sistema

Este apêndice descreve os passos necessários para a implantação do sistema, considerando os serviços utilizados, as configurações realizadas e os recursos necessários. O processo é simples e utiliza soluções modernas de hospedagem para *backend* e *frontend*, facilitando o acesso ao sistema por qualquer pessoa com conexão à internet.

1. Infraestrutura Utilizada

O sistema está dividido em duas partes principais: *backend* e *frontend*. A seguir estão os detalhes de cada parte e as ferramentas utilizadas para a implantação.

Backend

- Tecnologia: *ASP.NET Core*.
- Hospedagem: *Railway*.
- Banco de Dados: *PostgreSQL* (hospedado no *Railway*).
- Configuração:
  - O código do *backend* foi desenvolvido em *ASP.NET Core*, contendo todas as *APIs* necessárias para o funcionamento do sistema.
  - Após a conclusão do desenvolvimento, o código foi enviado ao *Railway*, que automatiza o processo de deployment.
  - O *Railway* gerou automaticamente uma base *URL*, acessível publicamente: *https://icorteapi.up.railway.app/*.
  - O banco de dados *PostgreSQL* foi configurado diretamente no Railway, com conexão já integrada ao *backend*.

Frontend

- Tecnologia: *React* com *Vite*.
- Hospedagem: *Netlify*.
- Configuração:

- O código *frontend* foi desenvolvido utilizando o framework *React* e o gerenciador de pacotes *Vite*, otimizando o desempenho e a velocidade de carregamento do sistema.
- Após o desenvolvimento, o projeto foi enviado ao *Netlify*, que também automatiza o processo de *deployment*.
- O *Netlify* gerou um link público para acesso ao sistema: *https://icorte.netlify.app/*.

2. Procedimentos para Implantação

Passo 1: Preparação do Backend

1. Crie uma conta gratuita no *Railway (https://railway.app)*.
2. Faça o upload do código do *backend* no *Railway*.
3. Configure as variáveis de ambiente necessárias para o funcionamento da *API* (como as strings de conexão do banco de dados).
4. Configure o *PostgreSQL* no *Railway* como um serviço adicional e integre-o ao *backend*.
5. Aguarde o processo de build e *deployment* ser concluído pelo *Railway*.
6. Após a finalização, a base *URL* será gerada automaticamente e estará disponível para uso.

Passo 2: Preparação do Frontend

1. Crie uma conta gratuita no *Netlify (https://www.netlify.com)*.
2. Faça o upload do código do *frontend* ou conecte um repositório *Git* (preferencialmente *GitHub*).
3. Configure o arquivo *vite.config.js* no *frontend* para apontar a base *URL* do *backend* hospedado no *Railway*.
4. Aguarde o processo de *build* e *deployment* ser concluído pelo *Netlify*.
5. Após a finalização, o link público de acesso ao sistema será gerado automaticamente.

3. Benefícios da Arquitetura Utilizada

Simplicidade: O processo de *deployment* é automatizado, reduzindo a necessidade de configurações manuais.

Escalabilidade: Tanto o *Railway* quanto o *Netlify* suportam crescimento do sistema, com opções de upgrade para planos pagos conforme a demanda.

Acessibilidade: O sistema está acessível por meio de *URLs* públicas, facilitando o acesso aos usuários.

Custo-Benefício: As ferramentas utilizadas oferecem planos gratuitos suficientes para projetos iniciais.

Com os passos acima concluídos, o sistema estará pronto para uso e poderá ser acessado por qualquer pessoa com acesso à internet.