Faculdade de Tecnologia de Americana "Ministro Ralph Biasi"

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Julio Cesar Ziviani

Rafael Cantovitz Bueno

Rogerio Henrique Bastos

BOOK SPACE:

Sistema web para gestão de espaços compartilhados

Americana, SP

2025

Julio Cesar Ziviani
Rafael Cantovitz Bueno
Rogerio Henrique Bastos

BOOK SPACE:

Sistema web para gestão de espaços compartilhados

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas na área de concentração em desenvolvimento de sistemas.

Orientador: Prof. Me. Thiago Salhab Alves

Este trabalho corresponde à versão final do Trabalho de Conclusão de Curso apresentado por Julio Cesar Ziviani, Rafael Cantovitz Bueno, Rogerio Henrique Bastos e orientado pelo Prof. Me. Thiago Salhab Alves.

Americana, SP
2025

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana Ministro Ralph Biasi- CEETEPS Dados Internacionais de Catalogação-na-fonte

ZIVIANI, Julio Cesar

BOOK SPACE: Sistema Web para Gestão de Espaços Compartilhados. / Julio Cesar Ziviani, Rafael Cantovitz Bueno, Rogerio Henrique Bastos – Americana, 2025.

129f.

Monografia (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana Ministro Ralph Biasi – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Thiago Salhab Alves

1. Desenvolvimento de software 2. Modelagem de sistemas 3. SQL – banco de dados. I. ZIVIANI, Julio Cesar, II. BUENO, Rafael Cantovitz, III. BASTOS, Rogerio Henrique IV. ALVES, Thiago Salhab V. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana Ministro Ralph Biasi

CDU: 681.3.05
681.5.01
681.3.07SQL

Elaborada pelo autor por meio de sistema automático gerador de ficha catalográfica da Fatec de Americana Ministro Ralph Biasi.

Julio Cesar Ziviani
Rafael Cantovitz Bueno
Rogerio Henrique Bastos

BOOK SPACE: Sistema Web para Gestão de Espaços Compartilhados

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas pelo Centro Paula Souza – FATEC Faculdade de Tecnologia de Americana Ministro Ralph Biasi.
Área de concentração: Análise e Desenvolvimento de Sistemas.

Americana, 23 de junho de 2025.

Banca Examinadora:

Thiago Salhab Alves
Mestre
Fatec Americana "Ministro Ralph Biasi"

José Luiz Zem
Doutor
Fatec Americana "Ministro Ralph Biasi"

Reydner Furtado Garbero
Doutor
Fatec Americana "Ministro Ralph Biasi"

DEDICATÓRIA

Dedicamos este trabalho, primeiramente, a Deus, por ter nos concedido força e sabedoria nos momentos de incerteza. Aos nossos pais, pelo amor incondicional, pelo apoio constante e por sempre acreditarem em nosso potencial, mesmo quando duvidamos de nós mesmos. À nossa família, pelo incentivo e pela compreensão ao longo de toda a jornada acadêmica. Aos amigos que estiveram presentes nos momentos mais desafiadores, oferecendo palavras de encorajamento e apoio sincero. E a todos que, de alguma forma, contribuíram para que este projeto se concretizasse.

RESUMO

O presente projeto apresenta o desenvolvimento de um aplicativo de gestão de espaços voltado para diferentes tipos de instituições, como escolas, universidades, empresas e órgãos públicos, que enfrentam dificuldades na administração de ambientes físicos compartilhados. Afinal, a prática recorrente de utilizar planilhas e e-mails para reservas compromete a eficiência institucional, gerando conflitos de agenda, baixa rastreabilidade e desperdício de tempo. Para solucionar isso, o sistema centraliza e automatiza o processo de reservas, permitindo que usuários previamente autorizados — como alunos, docentes, técnicos e colaboradores — visualizem a disponibilidade e realizem reservas com autonomia, agilidade e segurança. A aplicação foi desenvolvida com tecnologias modernas, incluindo Node.js, Next.js, TypeScript, TailwindCSS e Prisma ORM, utilizando PostgreSQL como banco de dados. O desenvolvimento seguiu a metodologia ágil Scrum, com papéis bem definidos, sprints quinzenais, organização das tarefas por meio de quadros no Trello e versionamento no GitHub. Entre as funcionalidades implementadas destacam-se o cadastro e gerenciamento de espaços, a visualização de disponibilidade, o controle de espaços de trabalho e permissões, além da criação de reservas. A segurança do sistema é assegurada por autenticação via JWT, criptografia de dados sensíveis e controle de acesso baseado em papéis (RBAC), garantindo que cada tipo de usuário possua permissões compatíveis com sua função. Além disso, a interface responsiva, acessível via navegador, possibilita que o aplicativo funcione em diferentes dispositivos, promovendo inclusão e eficiência. Sua arquitetura, feita de maneira modular, favorece a escalabilidade e a adaptação a variados contextos institucionais, além de permitir futuras expansões, como integração com calendários, envio de notificações automáticas e análises preditivas de uso. Dessa forma, a solução contribui diretamente para a organização e otimização do uso de espaços, potencializando a inteligência na gestão de ambientes e promovendo a transformação digital nas instituições.

PALAVRAS CHAVES: Reserva; Espaços; Gestão; Aplicação; Instituições.

ABSTRACT

This project presents the development of a space management application aimed at different types of institutions, such as schools, universities, companies and public institutions, which face difficulties in managing shared physical environments. After all, the recurring practice of using spreadsheets and emails for reservations compromises institutional efficiency, generating scheduling conflicts, low traceability and time wastage. To solve this, the system centralizes and automates the booking process, allowing previously authorized users - such as students, teachers, technicians and employees - to view availability and make reservations with autonomy, agility and security. The application was developed with modern technologies, including Node.js, Next.js, TypeScript, TailwindCSS and Prisma ORM, using PostgreSQL as database. The development followed the agile Scrum methodology, with well-defined roles, biweekly sprints, organization of tasks through boards in Trello and versioning on GitHub. Among the implemented functionalities stand out the registration and management of spaces, the visualization of availability, the control of workspaces and permissions, in addition to the creation of reservations. System security is ensured by authentication via JWT, sensitive data encryption and role-based access control (RBAC), ensuring that each type of user has permissions compatible with their role. In addition, the responsive interface, accessible via browser, allows the application to work on different devices, promoting inclusion and efficiency. Its architecture, made in a modular way, favors the scalability and adaptation to various institutional contexts, besides allowing future expansions, such as integration with calendars, sending automatic notifications and predictive analysis of use. Thus, the solution contributes directly to the organization and optimization of space use, enhancing intelligence in the management of environments and promoting digital transformation in institutions.

KEYWORDS: *Booking; Spaces; Management; Application; Institutions.*

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DD	Dicionário de dados
DER	Diagrama Entidade-Relacionamento
HTTP	HyperText Transfer Protocol
ID	Identificador
PNPM	Performant Node Package Manager
RBAC	Role-Based Access Control
SQL	Structured Query Language
UML	Unified Modeling Language

SUMÁRIO

1 INTRODUÇÃO

Diante das dificuldades frequentes de comunicação e coordenação na gestão de ambientes compartilhados, este projeto tem como objetivo desenvolver um aplicativo de gerenciamento, com o intuito de simplificar o processo de reserva de diferentes tipos de ambientes em instituições. A ideia surgiu a partir da observação de problemas frequentes, relacionados a reserva de salas para exames ou o agendamento de laboratórios de informática, que frequentemente geravam conflitos ou falhas na organização.

O aplicativo busca desenvolver uma plataforma que seja eficaz e intuitiva, permitindo os usuários consultarem a disponibilidade e realizar reservas de maneira simplificada e prática. Salas de aula, reunião, conferências, laboratórios, auditórios, áreas de recreação, espaços de estudo e ambientes de trabalho colaborativo, estão entre os ambientes abrangidos. A proposta visa solucionar problemas de comunicação e oferecer uma experiência otimizada para os usuários.

Quanto aos objetivos específicos, o trabalho inclui diversas metas que visam melhorar o gerenciamento de espaços em um ambiente institucional, sendo elas:

- Aplicar a metodologia Scrum durante o desenvolvimento do projeto;
- Facilitar a gestão de espaços dentro das empresas, garantindo maior organização e controle desses ambientes;
- Tornar o processo de agendamento de salas autônomo, visando maior eficiência e acessibilidade para os colaboradores;
- Contribuir para o crescimento da produtividade institucional, agilizando e otimizando seu processo interno.

O trabalho está organizado em cinco capítulos. Este primeiro capítulo trata-se da introdução, apresentando um contexto e os objetivos do projeto. No segundo capítulo, é discutido o projeto do sistema, detalhando as análises feitas em softwares com propósitos e características semelhantes ao sistema, com o intuito de entender as necessidades e expectativas do público-alvo. Também é abordado o levantamento de requisitos e a definição das tecnologias empregadas durante todo o projeto. O terceiro capítulo aborda a documentação do projeto, desenvolvida com o intuito de

facilitar o entendimento das funcionalidades do sistema, utilizando diagramas expressos na linguagem UML e quadros descritivos. No quarto capítulo é explorado as etapas do desenvolvimento do sistema utilizando a metodologia ágil Scrum, juntamente com as funções que cada membro desempenhou durante todo projeto. Também é apresentado a interface do sistema e seus recursos de tela. O quinto capítulo apresenta as considerações alcançadas ao longo do desenvolvimento do projeto, além de indicar sugestões e possibilidades para trabalhos futuros.

Por fim, logo após as referências bibliográficas, encontram-se os apêndices, que disponibilizam materiais complementares para uma melhor compreensão do projeto. O Apêndice A apresenta os recursos visuais, como um vídeo demonstrativo da aplicação e o *link* para o repositório do projeto. Já o Apêndice B reúne os quadros com a descrição dos métodos utilizados nos diagramas de classe.

2 PROJETO DO SISTEMA

Este capítulo tem como objetivo iniciar o desenvolvimento do sistema definindo seus requisitos e funcionalidades. Para isso, foram realizadas pesquisas em sites com aplicativos similares à ideia, visando refinar o projeto para apresentá-lo da forma mais completa possível.

2.1 Softwares Similares

Em relação aos aplicativos utilizados para a gestão de espaços, foram identificados diversos outros aplicativos com propósitos semelhantes ou similares. Entre aqueles atualmente mais utilizados, com características mais próximas do projeto, destacam-se: Escala, Calenda, Deskbee e Wiserxp.

2.1.1 Escala

O Escala é um aplicativo de organização de salas e mesas utilizado dentro de empresas para realizar a organização dentro de um escritório, permitindo a reserva de mesas e salas para utilização, como também a realização de reuniões. Durante a experiência, foi observado que há uma confusão na hora de criar cadastros de usuários dentro da rede de uma empresa. Também não é permitido que o usuário reserve o espaço por um determinado horário, apenas o dia completo. Há alguma duplicação de telas dentro do aplicativo, no entanto, a criação de espaços é clara e a reserva é eficiente, podendo limitar a quantidade de pessoas por reserva.

2.1.2 Calenda

Calenda é um aplicativo de gestão de salas descomplicada, feito para a reserva de salas de reunião de forma simples. Seu objetivo é desburocratizar as reservas de espaços dentro das empresas, permitindo um acompanhamento e

controle otimizados das reservas. Durante a análise, foi verificado que o aplicativo tem uma acessibilidade dificultada, pois é necessário agendar uma demonstração com a empresa para utilizá-lo. Entretanto, ele oferece diversos recursos adicionais, além de apresentar uma alta confiabilidade, demonstrada pela sua utilização por várias empresas já conhecidas no mercado.

2.1.3 Deskbee

Deskbee é um aplicativo de gerenciamento de espaços focado em aumentar a produtividade, tanto no espaço físico quanto no espaço digital de uma empresa. O site da empresa demonstra confiabilidade, com mais de 400 empresas já utilizando o serviço. Apresentando uma interface intuitiva, possibilitando a inclusão da planta do ambiente. Porém, foi encontrado alguns problemas, principalmente no site, onde múltiplos designs são apresentados para a mesma tela. Além disso, para acessar a demonstração do aplicativo, é necessário agendá-la, tornando a acessibilidade do aplicativo difícil.

2.1.4 Wiserxp

Wiserxp é um serviço de gerenciamento de salas de reunião que visa simplificar todo o processo de gestão de salas de reunião para empresas. O site demonstra alta confiabilidade, com várias empresas já utilizando seus aplicativos. Além de ser a ferramenta mais completa encontrada nessa pesquisa, oferecendo soluções abrangentes para os problemas das empresas. Possui múltiplos aplicativos para gerenciamento de espaço, tanto individual quanto para grandes empresas, e algumas ferramentas extras para facilitar o gerenciamento empresarial. Seu único problema é que é necessário agendar uma demonstração do aplicativo, o que torna o acesso complicado.

2.1.5 Resumo das comparações

Levando em consideração esse estudo e utilização dos aplicativos, foi elaborado o Quadro 1 demonstrando suas funcionalidades e a diferença entre os aplicativos analisados e o desenvolvido neste trabalho.

Quadro 1 - Comparativo de funcionalidades entre aplicativos similares e o aplicativo desenvolvido neste trabalho.

Funcionalidade	Escala	Deskbee	Calenda	Wiserxp	BookSpace
Criar, visualizar e gerenciar equipes	X		X		X
Adicionar membros	X	X	X	X	X
Definir permissões de acesso dos times a diferentes espaços	X				
Criar, editar e remover espaços	X	X	X	X	X
Anexar mapas ou layouts aos espaços	X	X	X	X	
Identificar posições pessoais nos mapas	X	X		X	
Limitar a quantidade de pessoas no espaço	X	X		X	X
Definir horários de acesso ao espaço	X	X	X	X	X
Adicionar atributos e recursos personalizados para cada espaço				X	X
Agendar reserva de espaço	X	X	X	X	X
Permitir reserva de posição por profissional em espaços	X	X	X	X	
Visualizar todas as reservas feitas		X	X	X	X
Realizar check-in nas reservas	X	X		X	

Fonte: Elaborado pelos autores (2025).

Os recursos selecionados para serem desenvolvidos no sistema foram definidos com base na análise de soluções similares, priorizando as funcionalidades mais relevantes e eficazes para o contexto institucional. A proposta busca unir, em uma única plataforma, aquilo que há de mais útil e funcional entre os concorrentes, oferecendo uma experiência otimizada ao usuário.

2.2 Levantamento de Requisitos

A engenharia de requisitos é o processo de descobrir, analisar, documentar e verificar requisitos de um sistema. Um requisito pode ser definido como uma descrição dos serviços fornecidos pelo sistema e as suas restrições operacionais (Sommerville, 2019). Tradicionalmente, os requisitos são divididos em dois tipos: requisitos funcionais e requisitos não funcionais.

2.2.1 Requisitos Funcionais

Os requisitos funcionais descrevem o que o sistema deve fazer, isto é, definem a funcionalidade desejada do software (Sommerville, 2019). O Quadro 2 apresenta os requisitos funcionais deste projeto.

Quadro 2 - Requisitos funcionais do projeto.

Identificação	Requisito Funcional	Prioridade
RF001	O usuário deve ser capaz de criar uma conta na plataforma.	Essencial
RF002	O usuário deve ser capaz de fazer login na plataforma utilizando as credenciais da sua conta cadastrada.	Essencial
RF003	O usuário deve ser capaz de gerenciar workspaces, incluindo criar, editar e excluir workspaces.	Essencial
RF004	O usuário deve ser capaz de adicionar e remover membros de uma workspace.	Essencial
RF005	O usuário deve ser capaz de definir permissões de acesso para os membros de uma workspace.	Importante
RF006	O usuário deve ser capaz de criar e gerenciar espaços na workspace, incluindo adicionar, editar e excluir espaços.	Essencial
RF007	O usuário deve ser capaz de limitar a quantidade de pessoas permitidas dentro de um espaço.	Desejável
RF008	O usuário deve ser capaz de definir horários de disponibilidade para os espaços na workspace	Importante
RF009	O usuário deve ser capaz de adicionar atributos e recursos específicos para cada espaço workspace.	Importante
RF010	O usuário deve ser capaz de sair de uma workspace.	Essencial
RF011	O usuário deve ser capaz de agendar reservas de espaço na plataforma.	Essencial
RF012	O usuário deve ser capaz de visualizar a disponibilidade dos espaços na plataforma.	Essencial
RF013	O usuário deve ser capaz de visualizar suas reservas.	Essencial

Fonte: Elaborado pelos autores (2025).

2.2.2 Requisitos Não Funcionais

“Os requisitos não funcionais são aqueles não diretamente relacionados às funções específicas fornecidas pelo sistema” (Sommerville, 2019). O Quadro 3 apresenta os requisitos não funcionais deste projeto.

Quadro 3 – Requisitos não funcionais do projeto.

Identificação	Requisito não funcional	Categoria	Prioridade
RNF001	A plataforma deve ser intuitiva e fácil de usar.	Usabilidade	Essencial
RNF002	A plataforma deve ser responsiva e compatível com uma variedade de dispositivos.	Usabilidade	Essencial
RNF003	O tempo de carregamento da plataforma deve ser rápido.	Desempenho	Essencial
RNF004	As senhas dos usuários devem ser criptografadas.	Segurança	Essencial
RNF005	Os formulários e campos de entrada na plataforma devem ser projetados de forma a minimizar erros de entrada e facilitar a entrada de dados pelos usuários	Usabilidade	Essencial

Fonte: Elaborado pelos autores (2025).

2.3 Recursos e Ferramentas

Esta seção contempla as tecnologias e ferramentas usadas nesse projeto, que foram selecionadas tendo em mente, compatibilidade, maturidade de mercado e alinhamento com práticas modernas de desenvolvimento web. A seguir, são apresentadas as principais escolhas técnicas, juntamente com as motivações, vantagens observadas e possíveis limitações encontradas ao longo do processo.

2.3.1 Ferramentas de apoio

Para dar suporte ao planejamento, modelagem e organização do projeto, tornou-se necessário adotar ferramentas que atendessem a critérios como compatibilidade com a *stack* adotada, suporte a metodologias ágeis, produtividade no

desenvolvimento e clareza na documentação. Nesse contexto, optou-se pelas seguintes ferramentas de apoio:

- StarUML: um modelador de software que busca dar suporte a modelagem ágil e concisa, tendo como principais recursos, diagramação de entidade-relacionamento, fluxo de dados, fluxograma, casos de uso, classes. (MKLABS CO.LTD, 2025)
- Visual Studio Code: é um editor de código gratuito, que possui suporte para depuração, execução de tarefas e controle de versão. Também conta com suporte integrado para JavaScript, TypeScript, Node.js e um ecossistema rico de extensões para outras linguagens. O objetivo desse editor é fornecer para o desenvolvedor as ferramentas para um ciclo rápido de código-compilação-depuração. (MICROSOFT, 2025)
- Trello: uma ferramenta de gerenciamento de projetos que auxilia na criação de planos, contribuição com projetos, organização de fluxos de trabalho e visualização de todo o progresso. (ATLASSIAN, 2025)

A adoção conjunta dessas ferramentas buscou formar uma base sólida de suporte à execução do projeto, unindo organização visual, documentação estruturada e um ambiente de desenvolvimento funcional e moderno.

2.3.2 Ambiente de Desenvolvimento

Para garantir um ambiente de desenvolvimento seguro, produtivo e alinhado com as boas práticas do desenvolvimento moderno, foram selecionadas ferramentas que oferecessem controle de tipo, eficiência na gestão de dependências e segurança na configuração do ambiente. As decisões a seguir refletem a busca por estabilidade, padronização e agilidade durante todas as etapas do projeto:

- Typescript: uma linguagem de programação fortemente tipada que adiciona uma nova sintaxe ao Javascript para uma integração mais estrita com o editor, podendo detectar erros no próprio editor. (OPENJS FOUNDATION, 2025)

- Pnpm: um gerenciador de pacotes Javascript que busca rapidez e eficiência. Sendo duas vezes mais rápido do que o npm, o pnpm utiliza links de hard links para armazenar os pacotes, assim economizando espaço no disco e tornando mais ágil a instalação e manutenção das dependências. (CONTRIBUTORS OF PNPM, 2025)
- Dotenv: um modulo independente que carrega variáveis de ambiente de um arquivo .env para process.env. O modulo utilizou como base a metodologia aplicada no aplicativo Twelve-Factor, onde é separado do código todo armazenamento da configuração no ambiente. (DOTENVX, 2025)

A escolha do Typescript visou aumentar a previsibilidade do código e reduzir falhas durante o desenvolvimento, graças à sua integração com o Visual Studio Code, torna-se possível detectar erros durante a escrita, proporcionando maior confiança durante o desenvolvimento. Para o gerenciamento de pacotes, adotou-se o Pnpm devido ao seu desempenho superior, uso eficiente de cache e suporte nativo a monorepos, o que o tornou especialmente vantajoso em projetos com múltiplos pacotes e ciclos de atualização constantes. Já o Dotenv foi utilizado para garantir uma configuração segura e desacoplada do código-fonte, em conformidade com boas práticas de implantação. Em conjunto, essas ferramentas formaram a base do ambiente de desenvolvimento, promovendo padronização, segurança e eficiência no processo de construção do sistema.

2.3.3 Versionamento

Para garantir controle, rastreabilidade e colaboração durante o desenvolvimento do sistema, foi necessário adotar ferramentas que permitissem gerenciar versões de forma eficiente e suportassem o trabalho em equipe de maneira segura e integrada. As seguintes ferramentas foram escolhidas com esse propósito:

- Git: um software de controle de versão, projetado para lidar com projetos pequenos e grandes de maneira rápida e eficiente, tendo funções como

ramificação, áreas de preparação e fluxo de trabalho centralizado. (SOFTWARE FREEDOM CONSERVANCY, 2025)

- Github: uma plataforma baseada em nuvem onde é possível, armazenar, compartilhar códigos e trabalhar com outras pessoas. (GITHUB, 2025)

A escolha do Git baseou-se em sua robustez no controle de versões e na flexibilidade para gerenciar ramificações em projetos colaborativos. Como complemento, adotou-se o GitHub para hospedagem do repositório e integração com fluxos de trabalho ágeis. Juntas, essas ferramentas garantiram versionamento seguro, colaboração eficiente e escalabilidade para as futuras etapas do projeto.

2.3.4 Front-end

A definição das tecnologias de *front-end* foi orientada por critérios como reutilização de componentes, performance, acessibilidade, escalabilidade e compatibilidade com a arquitetura moderna adotada no projeto. A seguir, apresentam-se as ferramentas adotadas:

- ReactJS: uma biblioteca Javascript para a criação de interfaces de usuário web e nativas a partir de componentes, que são reutilizáveis e responsivos a interação do usuário. (META OPEN SOURSE, 2025)
- NextJS: um framework React para construir aplicativos web-full-stack, enquanto utiliza os componentes React para construir interfaces de usuário, o Next.js faz otimizações e proporciona novos recursos. (VERCEL, 2025)
- Radix UI: uma biblioteca que fornece componentes React sem estilo com foco na construção de interfaces de usuário de maneira acessível e flexível. (WORKOS, 2025)
- Tailwind CSS: um framework CSS com foco na utilidade que fornece várias classes utilitárias, como flex, text-center e rotate-90 que podem ser compostas para construir qualquer design. (WATHAN, 2025)
- PostCSS: uma ferramenta para transformar estilos CSS com plugins feitos em Javascript, ideal para realizar tarefas de modificação,

otimização ou correção de erros no CSS de maneira automatizada. (REVELO, 2025)

- Phosphor Icons: uma coleção de ícones para interfaces, diagramas e apresentações. (ZHANG, FRIED, 2025)
- React Hook Form: uma biblioteca para gerenciamento de formulários que permite de maneira simplificada criar formulários de alto desempenho, flexíveis, extensíveis e de fácil validação. (BEEKAI, 2025)

2.3.5 Back-end

A definição das tecnologias de *back-end* considerou aspectos como compatibilidade com o *front-end*, desempenho, segurança e facilidade de manutenção. A escolha das ferramentas buscou reduzir a curva de aprendizado da equipe e promover uma arquitetura leve e escalável, alinhada aos princípios do desenvolvimento ágil. As ferramentas selecionadas forma:

- NodeJS: um ambiente de execução Javascript, que permite aos desenvolvedores criarem servidores e aplicativos web rápidos e escalonáveis. (OPENJS FOUNDATION, 2024)
- Fastify: um framework web de alto desempenho, que permite criar um servidor eficiente, sendo altamente focado em fornecer a melhor experiência ao desenvolvedor com o mínimo de sobrecarga e uma arquitetura de plugins poderosa. (FASTIFY, 2025)
- Zod: uma biblioteca de validação de tipos de dados que engloba desde tipos de dados mais simples como *string* até tipos de dados mais complexos. (MCDONNELL, 2025)
- Bcrypt: uma biblioteca que auxilia na criptografia de senhas para serem armazenadas em um banco de dados. A biblioteca se destaca por contar com funções adaptativas que tornam a contagem de iterações mais lentas fornecendo maior segurança e por utilizar o *salt,* um valor aleatório adicionado a senha antes de ser criptografado. (MAHAPATRA, 2025)

- Jsonwebtoken: uma biblioteca que permite realizar a troca de informações entre partes como um objeto JSON e garante que elas não foram alteradas. (LACEY, 2025)

Essas decisões permitiram construir uma base de *back-end* eficiente, segura e fortemente integrada ao restante da aplicação, com foco em produtividade, performance e escalabilidade. A compatibilidade entre as ferramentas, aliada à sua leveza e maturidade, foi essencial para o sucesso do projeto.

2.3.6 Banco de Dados

A camada de persistência de dados foi estruturada com foco em robustez, consistência e integração eficiente com o ambiente Typescript adotado no projeto. Sendo escolhido:

- PostgreSQL: um sistema de banco de dados relacional que combina linguagem SQL com diversos recursos que buscam atender às necessidades específicas dos desenvolvedores. (POSTGRESQL, 2025)
- Prisma: é uma ferramenta de banco de dados que simplifica a interação com o banco de dados, sendo ideal para a construção de aplicativos que irão necessitar de um uso intensivo de dados. A ferramenta se destaca por possuir uma api de consulta tipada e suporte a múltiplos bancos de dados. (PRISMA, 2025)

2.3.7 Ferramentas para qualidade de código

Para garantir consistência, legibilidade e segurança na base de código ao longo do desenvolvimento, foi adotado um conjunto de ferramentas voltado à análise estática, formatação automatizada, validação de *commits* e testes automatizados. Essas decisões refletem a busca por um ambiente de desenvolvimento limpo, previsível e alinhado com as boas práticas:

- Eslint: uma ferramenta que analisa de maneira estática o código Javascript, a fim de identificar e corrigir problemas, como erros de sintaxe e problemas de estilo, seguindo regras e práticas definidas. (OPENJS FOUNDATION, 2025)
- Prettier: um formatador de código com suporte a Javascript, Typescript JSX, CSS, HTML, entre outras linguagens. Ele pode remover o estilo original e certificar que todo código esteja seguindo um estilo consistente e legível. (PRETTIER, 2025)
- Husky: uma biblioteca que permite automatizar tarefas ao realizar ações com o Git, tornando possível formatar o código e executar testes durante um *commit* ou enquanto envia o código para o repositório. (HUSKY, 2025)
- Lint-staged: uma biblioteca que permite executar analisadores de códigos em arquivos na área de preparação do Git, antes do *commit*. (JÄPPINEN. Iiroj, 2025)
- Git-commit-msg-linter: uma biblioteca que compara a mensagem de *commit* do Git com uma diretriz de mensagens de *commit*, assim garantindo que toda mensagem seja válida se estiver de acordo com as diretrizes, caso não for validado, o *commit* é abortado. (LEGEND80s, 2024)
- Jest: um framework de testes focado em trazer simplicidade e facilidade de uso na correção de qualquer código Javascript. Os testes são feitos em seus próprios processos e são executados em conjunto maximizando o desempenho. (OPENJS FOUNDATION, 2025)

Ao adotar esse conjunto de ferramentas, o projeto passou a contar com uma camada robusta de proteção contra erros triviais, além de promover consistência e transparência no fluxo de trabalho. A qualidade do código deixou de ser uma preocupação apenas no momento da entrega e passou a ser garantida continuamente ao longo de toda a construção do sistema.

3 MODELAGEM

Este capítulo tem como objetivo a documentação do projeto com o intuito de simplificar o entendimento das funcionalidades do projeto, com a utilização de diagramas de caso de uso, e quadros para detalhar a usabilidade do sistema. A documentação está expressa na linguagem *UML* para modelar os casos de uso e o diagrama de classe.

3.1 Casos De Uso

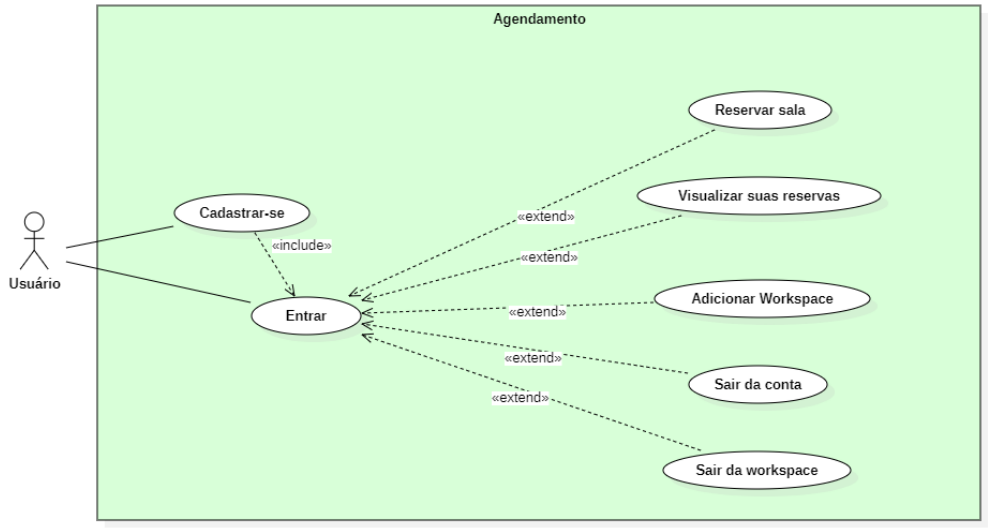
Os diagramas de caso de uso descrevem um cenário de funcionalidades do ponto de vista do usuário, catalogando os requisitos funcionais do sistema. Dentro do diagrama são retratados os atores (representado pelos bonecos), as funcionalidades (representadas pelos balões com a ação escrita por dentro) e as relações (representadas pelas linhas).

Os atores que interagem com o sistema são: o usuário, o gestor da *workspace*, o dono de *workspace*. O sistema é um caso de uso explícito e se trata do sistema em si em que os casos de uso acontecem.

- **Usuário** é o ator que representa os utilizadores deste aplicativo. Ele pode criar conta, entrar no aplicativo, sair da conta, reservar sala, visualizar suas reservas e adicionar uma *workspace*, tornando-se dono da *workspace*.
- **Gestor da Workspace** é o ator que representa o usuário após ser adicionado como um gestor pelo dono. Ele pode adicionar Usuários normais, remover usuários, adicionar espaços, editar espaços e remover espaços.
- **Dono da workspace** é o ator que representa que o usuário após a criação de uma *workspace*, tendo permissões específicas sobre a *workspace*. Ele pode editar a *workspace*, deletar *workspace*, adicionar gestores, e executar as outras funções disponíveis para os gestores.

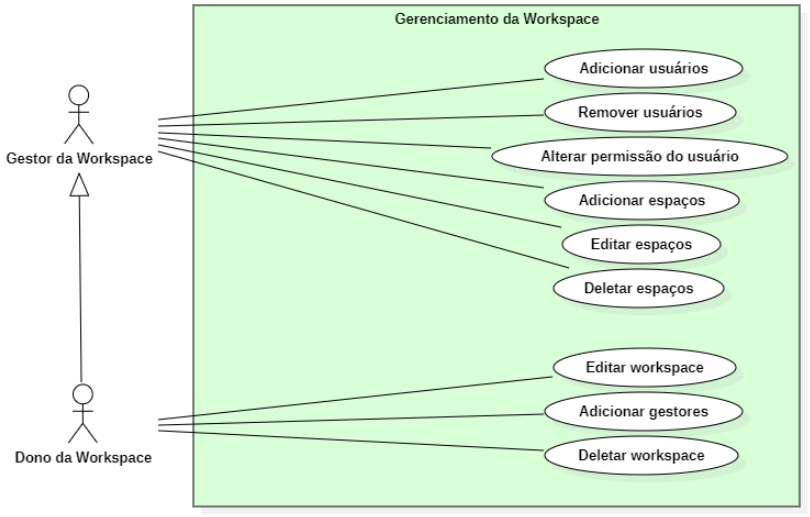
A Figura 1 apresenta o caso de uso para a entrada do usuário no sistema e as principais ferramentas do sistema. Enquanto a Figura 2 apresenta o gerenciamento da *workspace* feito pelo dono e pelo gestor da *workspace*.

Figura 1 - Diagrama de caso de uso do usuário utilizando o sistema



Fonte: Elaborado pelos autores (2025).

Figura 2 – Diagrama de caso de uso do gerenciamento da Workspace



Fonte: Elaborado pelos autores (2025).

3.2 Documentação dos Casos de Uso

Cada funcionalidade dos diagramas de casos de uso será descrita do Quadro 4 ao Quadro 19.

Quadro 4 - Caso de uso "Cadastrar-se"

Nome do caso de uso	Cadastrar-se
Atores envolvidos	Usuário
Objetivo	Este caso de uso descreve os passos do cadastro das informações do usuário
Ações do ator	**Ações do Sistema**
1. O usuário informa os dados necessários para o cadastro (Nome, e-mail e senha) e clicar em “Cadastrar-se”.	
	2. A aplicação web válida as informações.
	3. A aplicação web se comunica com a API.
	4. A API válida novamente as informações e verifica se o e-mail está disponível.
	5. A API registra o usuário.
	6. A API retorna uma resposta de sucesso.
	7. A aplicação web realiza o login.
	8. A aplicação web redireciona para a tela de início.
Validações	Deve ser informado o nome, e-mail e senha, o cadastro não é realizado quando o e-mail já existe na base de dados.

Fonte: Elaborado pelos autores (2025).

Quadro 5 – Caso de uso “Entrar”.

Nome do caso de uso	Entrar
Atores envolvidos	Usuário
Objetivo	Este caso de uso descreve os passos do login do usuário
Ações do ator	**Ações do Sistema**
1. O usuário informa os dados necessários para entrar (E-mail e senha) e clicar em “Entrar”.	
	2. A aplicação web comunica-se com a API.
	3. A API válida as informações.
	4. A API verifica se as credenciais estão corretas
	5. A API autêntica o usuário na base de dados
	6. A API retorna um token de acesso e as informações do usuário
	7. Após a autenticação, a aplicação web salva essas informações em memória.
	8. A Aplicação web redireciona para a página inicial do aplicativo
Validações	Para o login o usuário deve informar seu e-mail e senha, para que seja autenticado essas informações devem ser compatíveis com a base de dados.

Fonte: Elaborado pelos autores (2025).

Quadro 6 – Caso de uso “Sair da conta”.

Nome do caso de uso	Sair da conta
Atores envolvidos	Usuário
Objetivo	Este caso de uso descreve os passos para o usuário sair do sistema
Ações do ator	**Ações do Sistema**
1. O usuário clica em “Sair”	
	2. A aplicação web comunica-se com a API.
	2. A API remove o token do banco de dados
	3. A API retorna uma resposta de sucesso
	4. A aplicação web remove o token da memória
Validações	

Fonte: Elaborado pelos autores (2025).

Quadro 7 – Caso de uso “Reservar sala”.

Nome do caso de uso	Reservar sala
Atores envolvidos	Usuário
Objetivo	Este caso de uso descreve os passos para reservar uma sala.
Ações do ator	**Ações do Sistema**
1. O usuário após realizar o login no sistema clica em “Agendar”	
	2. A aplicação web abre um formulário onde o usuário deve inserir as informações da reserva (dia da reserva, hora de início, hora do final).
3. O usuário informa as informações requeridas	
	4. A aplicação web válida os dados.
	5. A aplicação web comunica-se com a API.
	6. A API válida os dados novamente.
	7. A API verifica se a reserva é válida.
	8. A API registra os dados no banco de dados.
	9. A API retorna as informações da reserva.
Validações	Para criar uma reserva o usuário deve estar autenticado.

Fonte: Elaborado pelos autores (2025).

Quadro 8 – Caso de uso “Visualizar suas reservas”.

Nome do caso de uso	Visualizar suas reservas
Atores envolvidos	Usuário
Objetivo	Este caso de uso descreve os passos para um usuário visualizar suas reservas.
Ações do ator	**Ações do Sistema**
1. O usuário após realizar o login no sistema clica em “Agendamentos”	
	2. A aplicação web comunica-se com a API
	3. A API retornar as reservas do usuário.
	4. A aplicação web atualiza a lista de reservas.
Validações	Para listar as reservas o usuário deve estar autenticado.

Fonte: Elaborado pelos autores (2025).

Quadro 9 – Caso de uso “Adicionar Workspace”.

Nome do caso de uso	Adicionar Workspace
Atores envolvidos	Usuário
Objetivo	Este caso de uso descreve os passos da criação de uma Workspace
Ações do ator	**Ações do Sistema**
1. O usuário após realizar o login no sistema clica em “Adicionar Workspace”	
	2. A aplicação web abre um formulário onde o usuário deve inserir as informações da *workspace* (nome, tag).
3. O usuário informa as informações requeridas	
	4. A aplicação web válida os dados.
	5. A aplicação web comunica-se com a API.
	6. A API válida os dados novamente.
	7. A API registra os dados no banco de dados.
	8. A API retorna as informações da *workspace*.
	9. A aplicação web atualiza a lista de *workspace*.
Validações	Para criar uma *workspace* o usuário deve estar autenticado.

Fonte: Elaborado pelos autores (2025).

Quadro 10 – Caso de uso “Sair da Workspace”.

Nome do caso de uso	Sair da Workspace
Atores envolvidos	Usuário
Objetivo	Esse caso de uso descreve os passos de um usuário para sair de uma *workspace* na qual ele seja membro.
Ações do ator	**Ações do Sistema**
1. O usuário após realizar o login no sistema seleciona a *workspace* na qual ele deseja sair.	
2. O usuário clicar em “Sair da Workspace”.	
	3. A aplicação Web comunica-se com a API.
	4. A API verifica se o usuário possuí permissão para sair da *workspace*.
	5. A API remove o usuário da *workspace*.
	6. A API retorna uma resposta de sucesso.
	7. A aplicação web atualiza a lista de *workspaces*.
Validações	Para sair de uma *workspace* o usuário não pode ser o dono dela.

Fonte: Elaborado pelos autores (2025).

Quadro 11 – Caso de uso “Adicionar Usuário”.

Nome do caso de uso	Adicionar usuário
Atores envolvidos	Gestor e dono da Workspace
Objetivo	Esse caso de uso descreve os passos da adição de um usuário na *workspace*
Ações do ator	**Ações do Sistema**
1. O usuário após realizar o login no sistema clica em “Adicionar usuário”.	
	2. A aplicação web abre um formulário onde o usuário deve inserir as informações do usuário (nome, cargo - Padrão).
3. O usuário insere as informações solicitadas e clica em adicionar usuário	
	4. A aplicação web faz a validação das informações.
	5. A aplicação web comunica-se com a API
	6. A API faz a validação das informações novamente.
	7. A API verifica se o usuário possuí permissão para adicionar um usuário à *workspace*.
	8. A API retorna as informações do usuário.
	9. A aplicação web atualiza a lista de usuários da *workspace*.
Validações	Para adicionar um usuário a uma *workspace* o usuário deve estar autenticado e ser o dono ou um gestor da *workspace*

Fonte: Elaborado pelos autores (2025).

Quadro 12 – Caso de uso “Alterar permissão do usuário”.

Nome do caso de uso	Alterar permissão do usuário
Atores envolvidos	Gestor e Dono da Workspace
Objetivo	Esse caso de uso descreve os passos de um usuário para mudar as permissões de um usuário dentro de uma *workspace* a qual ele é gerente ou dono
Ações do ator	**Ações do Sistema**
1. O usuário após realizar o login no sistema seleciona a *workspace* na qual vai fazer a alteração	
2. O usuário seleciona o usuário no qual será mudada a permissão	
3. O usuário informa para qual permissão será alterada	
	4. A aplicação web comunica-se com a API.
	5. A API válida as informações.
	6. A API verifica se o usuário possuí permissão para alterar a permissão do usuário.
	7. A API muda as permissões do usuário.
	8. A API retorna uma resposta de sucesso.
	9. A aplicação web atualiza a lista de usuários.
Validações	O sistema valida se o usuário é gerente ou dono da *workspace* para verificar se ele pode alterar as permissões. Além disso, um gerente não pode alterar o cargo para igual ou maior que o dele, e o dono não pode adicionar outro dono.

Fonte: Elaborado pelos autores (2025).

Quadro 13 – Caso de uso “Remover usuário”.

Nome do caso de uso	Remover usuário
Atores envolvidos	Gestor e dono da Workspace
Objetivo	Esse caso de uso descreve os passos de um usuário para remover um usuário da *workspace*.
Ações do ator	**Ações do Sistema**
1. O usuário seleciona a *workspace* na qual vai remover o usuário.	
2. O usuário clica em “Remover usuário”.	
	3. A aplicação web comunica-se com a API.
	4. A API verifica se o usuário possuí permissão para remover um usuário da *workspace*.
	5. A API remove o usuário.
	7. A API retorna uma resposta de sucesso.
	8. A aplicação web atualiza a lista de usuários.
Validações	O sistema valida se o usuário é gerente ou dono da *workspace* para verificar se ele pode remover um usuário.

Fonte: Elaborado pelos autores (2025).

Quadro 14 – Caso de uso “Adicionar espaço”.

Nome do caso de uso	Adicionar espaço
Atores envolvidos	Gestor e dono da Workspace
Objetivo	Esse caso de uso descreve os passos de um usuário adicionar um espaço à *workspace*.
Ações do ator	**Ações do Sistema**
1. O usuário seleciona a *workspace* na qual vai adicionar o espaço.	
2. O usuário clica em “Adicionar espaço”.	
	3. A aplicação web abre um formulário onde o usuário deve inserir as informações do espaço (nome, descrição, quantidade de pessoas, recursos e a disponibilidade).
4. O usuário insere as informações solicitadas e clica em “Adicionar espaço”.	
	5. A aplicação web válida as informações.
	6. A aplicação web comunica-se com a API.
	7. A API faz válida as informações novamente.
	8. A API verifica se o usuário possuí permissão para adicionar um espaço na *workspace*.
	9. A API registra as informações do espaço.
	10. A API retornar as informações do espaço.
	11. A aplicação web atualiza a lista de espaços.
Validações	O sistema valida se o usuário é gerente ou dono da *workspace* para verificar se ele pode adicionar um espaço.

Fonte: Elaborado pelos autores (2025).

Quadro 15 – Caso de uso “Editar espaço”.

Nome do caso de uso	Editar espaço
Atores envolvidos	Gestor e dono da Workspace
Objetivo	Esse caso de uso descreve os passos de um usuário para editar um espaço da *workspace*.
Ações do ator	**Ações do Sistema**
1. O usuário seleciona a *workspace* na qual vai editar o espaço.	
2. O usuário clica em “Editar espaço”.	
	3. A aplicação web abre um formulário onde o usuário deve inserir as informações do espaço (nome, descrição, quantidade de pessoas, recursos e a disponibilidade).
4. O usuário insere as informações solicitadas e clica em “Editar espaço”.	
	5. A aplicação web válida as informações.
	6. A aplicação web comunica-se com a API.
	7. A API faz válida as informações novamente.
	8. A API verifica se o usuário possuí permissão para adicionar um espaço na *workspace*.
	9. A API registra as novas informações do espaço.
	10. A API retornar as informações do espaço.
	11. A aplicação web atualiza a lista de espaços.
Validações	O sistema valida se o usuário é gerente ou dono da *workspace* para verificar se ele pode editar um espaço.

Fonte: Elaborado pelos autores (2025).

Quadro 16 – Caso de uso “Deletar espaço”.

Nome do caso de uso	Deletar espaço
Atores envolvidos	Gestor e dono da *Workspace*
Objetivo	Esse caso de uso descreve os passos de um usuário para deletar um espaço da *workspace*.
Ações do ator	**Ações do Sistema**
1. O usuário seleciona a *workspace* na qual vai deletar o espaço.	
2. O usuário clica em “Deletar espaço”.	
	3. A aplicação web comunica-se com a API.
	4. A API verifica se o usuário possuí permissão para deletar um espaço na *workspace*.
	5. A API remove os itens relacionados ao espaço.
	6. A API remove o espaço.
	7. A API retorna uma resposta de sucesso.
	8. A aplicação web atualiza a lista de espaços.
Validações	O sistema valida se o usuário é gerente ou dono da *workspace* para verificar se ele pode deletar um espaço.

Fonte: Elaborado pelos autores (2025).

Quadro 17 – Caso de uso “Adicionar gestor”.

Nome do caso de uso	Adicionar gestor
Atores envolvidos	Dono da Workspace
Objetivo	Esse caso de uso descreve os passos da adição de um gestor na *workspace*
Ações do ator	**Ações do Sistema**
1. O usuário após realizar o login no sistema clica em “Adicionar usuário”.	
	2. A aplicação web abre um formulário onde o usuário deve inserir as informações do usuário (nome, cargo - Gerente).
3. O usuário insere as informações solicitadas e clica em “Adicionar usuário”.	
	4. A aplicação web faz a validação das informações.
	5. A aplicação web comunica-se com a API
	6. A API faz a validação das informações novamente.
	7. A API verifica se o usuário possuí permissão para adicionar um gestor à *workspace*.
	8. A API retorna as informações do usuário.
	9. A aplicação web atualiza a lista de usuários da *workspace*.
Validações	Para adicionar um gestor a uma *workspace* o usuário deve estar autenticado e ser o dono da *workspace*.

Fonte: Elaborado pelos autores (2025).

Quadro 18 – Caso de uso “Editar Workspace”.

Nome do caso de uso	Editar Workspace
Atores envolvidos	Dono da Workspace
Objetivo	Este caso de uso descreve os passos da edição de uma *workspace*
Ações do ator	**Ações do Sistema**
1. O usuário após realizar o login no sistema clica em “Editar Workspace”	
	2. A aplicação web abre um formulário onde o usuário deve inserir as informações da *workspace* (nome, tag).
3. O usuário realiza as mudanças e clica em “Editar Workspace”.	
	4. A aplicação web válida os dados.
	5. A aplicação web comunica-se com a API.
	6. A API válida os dados novamente.
	7. A API verifica se o usuário possuí permissão para alterar os dados da *workspace*.
	8. A API registra os novos dados no banco de dados.
	9. A API retorna as informações da *workspace*.
	10. A aplicação web atualiza a lista de *workspace*.
Validações	Para editar uma *workspace* o usuário deve estar autenticado e ser dono da *workspace*.

Fonte: Elaborado pelos autores (2025).

Quadro 19 – Caso de uso “Deletar Workspace”.

Nome do caso de uso	Deletar Workspace
Atores envolvidos	Dono da Workspace
Objetivo	Este caso de uso descreve os passos de deleção de uma *workspace*.
Ações do ator	**Ações do Sistema**
1. O usuário após realizar o login no sistema clica em “Excluir Workspace”.	
	2. O aplicativo web comunica-se com a API
	3. A API verifica se o usuário possuí permissão de exclusão da *workspace*.
	4. A API remove os usuários da *workspace*, espaços e outras informações da *workspace*.
	5. A API remove a *workspace*.
	6. A API retorna resposta de sucesso.
	7. O aplicativo web atualiza a lista de *workspaces*.
Validações	Para remover uma *workspace* o usuário deve estar autenticado e ser dono da *workspace*.

Fonte: Elaborado pelos autores (2025).

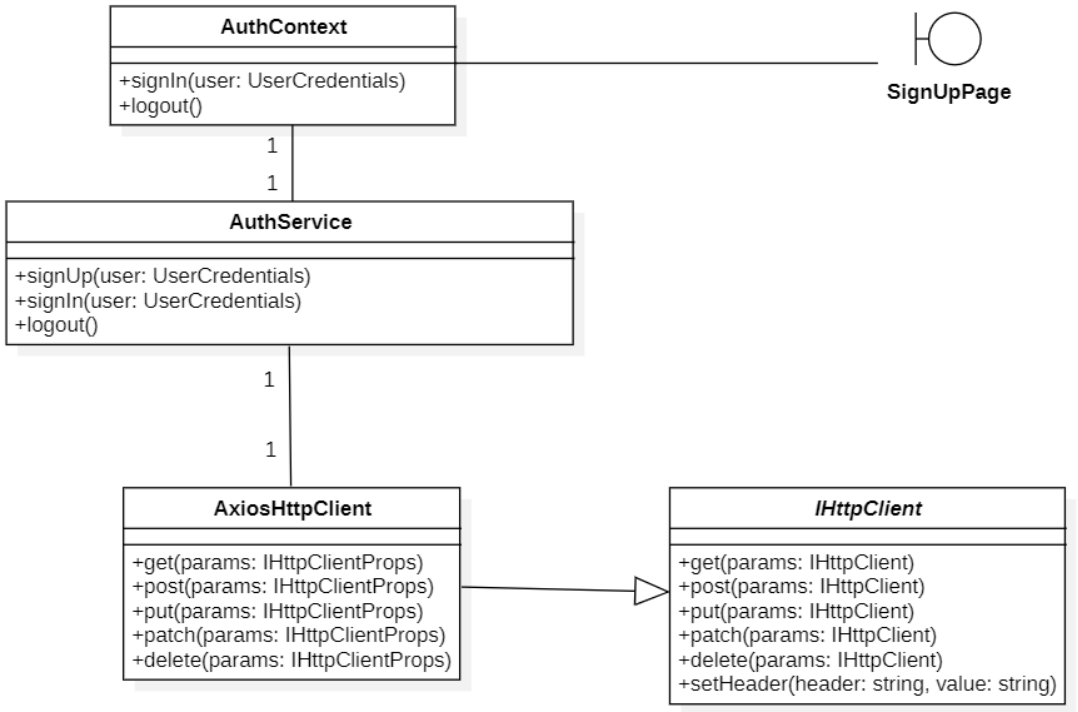
3.3 Diagrama de Classe da Aplicação Web

A arquitetura do projeto foi dividida em duas partes distintas: *front-end* (aplicação web) e *back-end* (API). Essa separação permite uma clara distinção entre a interface do usuário e a lógica do servidor, facilitando a manutenção, a escalabilidade e a distribuição de responsabilidades entre diferentes equipes. Tal decisão arquitetural foi adotada com base em critérios técnicos e operacionais que buscam otimizar o desenvolvimento, a organização e a segurança do sistema. Ao isolar a camada de apresentação da lógica de negócios e das operações com banco de dados, torna-se possível evoluir e manter cada parte de forma independente, minimizando impactos colaterais.

Essa abordagem também favorece a especialização das equipes, o reaproveitamento da API por múltiplos canais (como aplicações mobile), e a escalabilidade individual de cada serviço conforme a demanda. Além disso, concentra a lógica sensível no *back-end*, reduzindo a exposição de dados no cliente e fortalecendo a segurança. Com isso, a divisão entre *front-end* e API se mostra não apenas uma escolha estrutural, mas uma estratégia de design que contribui diretamente para a robustez, eficiência e longevidade da aplicação.

A partir dessa arquitetura bem definida, o *front-end* da aplicação web foi organizado em seis páginas principais, cuja lógica é encapsulada em *hooks* e *contexts* personalizados. Esses componentes interagem com a API por meio de serviços especializados que utilizam um cliente HTTP, garantindo uma comunicação clara e modular entre as camadas. A estrutura detalhada dessa organização é ilustrada nos diagramas de classe, que representam tanto a disposição das entidades e seus relacionamentos quanto as operações acionadas pelos usuários, servindo de base para o modelo lógico do sistema.

Figura 3 - Diagrama da página de cadastro.

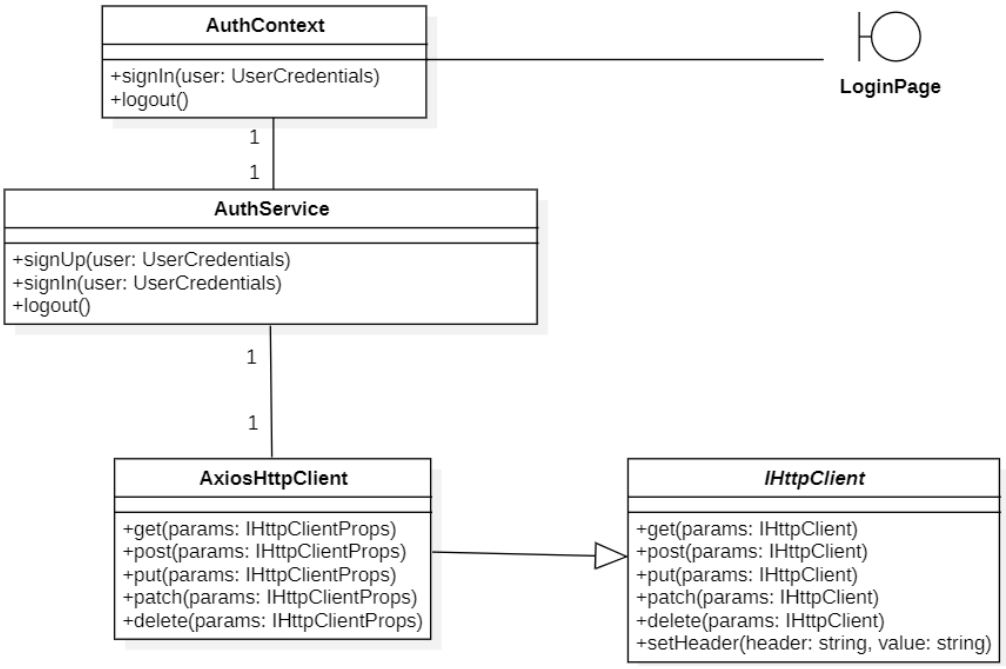


Fonte: Elaborado pelos autores (2025).

A página de cadastro, evidenciada na Figura 3 permite que novos usuários criem suas contas, usando para isso três classes auxiliares e uma interface, sendo as respectivas:

- *AuthContext:* Está classe tem como objetivo por gerenciar o estado de autenticação da aplicação (Figura 3). Ela fornece um contexto para que diferentes partes da aplicação possam acessar e modificar o estado de autenticação de forma centralizada e consistente.
- *AuthService:* Está classe tem como objetivo é responsável por lidar com a lógica de autenticação propriamente dita (Figura 3). Ela interage com a API de autenticação para realizar operações como cadastro, *login*, *logout*.
- *AxiosHttpClient:* Está classe tem como objetivo é uma implementação específica de um cliente HTTP utilizando a biblioteca Axios (Figura 3).
- *IHttpClient*: Uma interface com o objetivo de definir um contrato para a implementação de clientes HTTP. Ela especifica os métodos que qualquer implementação de cliente HTTP deve fornecer (Figura 3).

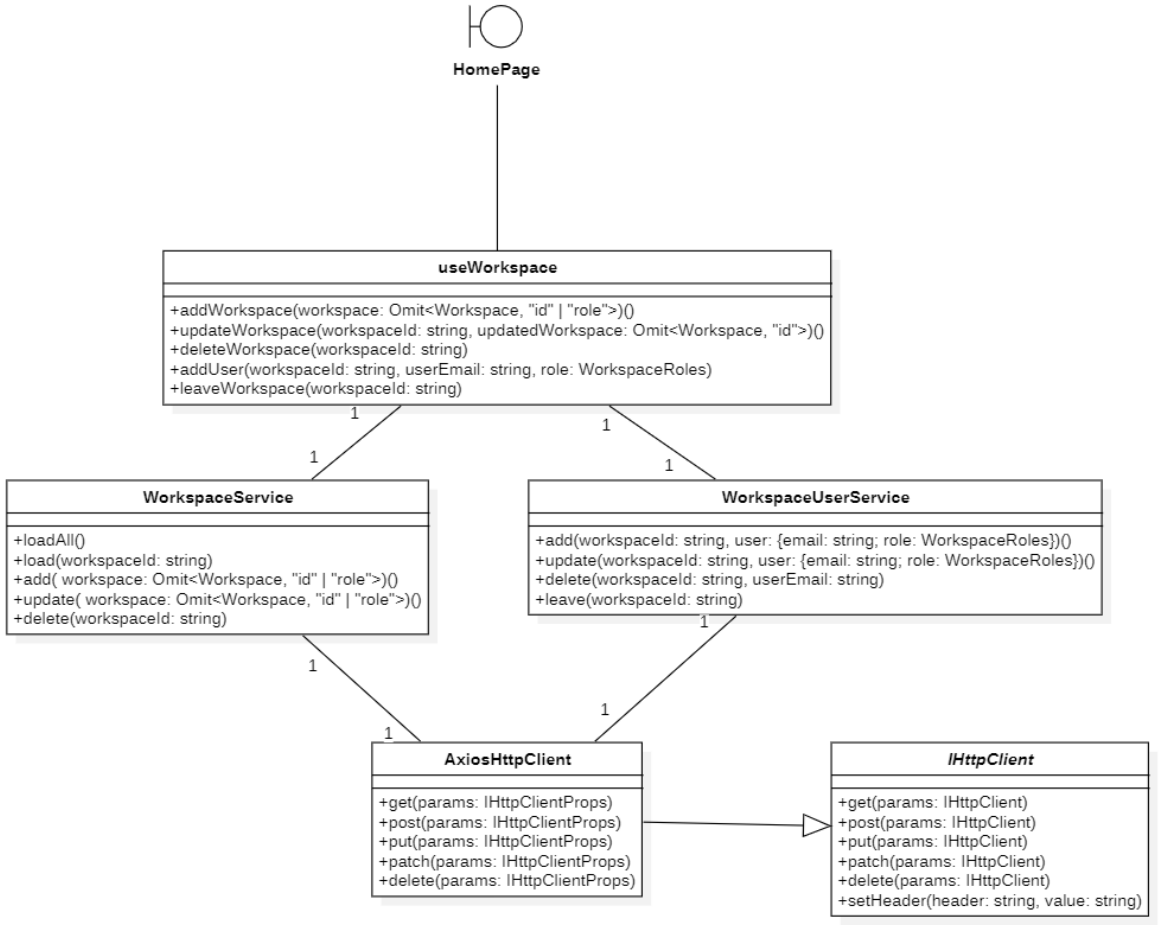
Figura 4 - Diagrama da página de Login.



Fonte: Elaborado pelos autores (2025).

A página de login, permite que os usuários acessem a conta deles (Figura 4). Para isso, usa três classes, já vistas e explicadas anteriormente na Figura 3 – *AuthContext* no Quadro 28, *AuthService* no Quadro 29 e *AxiosHttpClient* no Quadro 30.

Figura 5 - Diagrama da página inicial

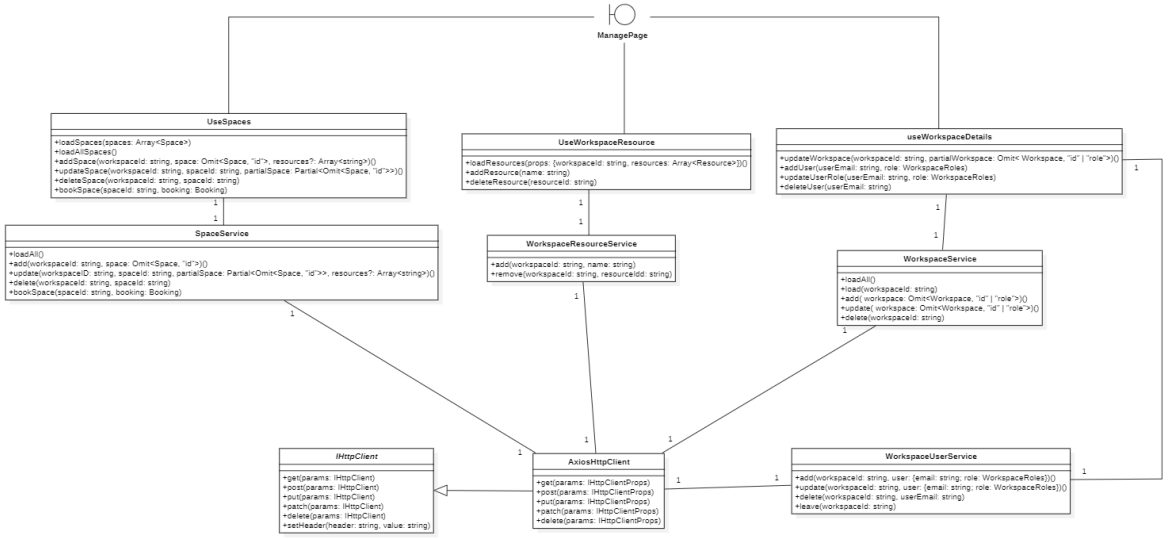


Fonte: Elaborado pelos autores (2025).

A página de início, reúne um conjunto de *workspaces*, onde o usuário pode adicionar *workspace* e realizar outras funções simples (Figura 5). Para isso, utiliza de quatro classes, note que, o *AxiosHttpClient* foi explicado no Quadro 30, porém as outras classes são:

- *useWorkspace*: Está classe tem como objetivo gerenciar e fornecer o estado e as operações relacionadas a uma *workspace* dentro de uma aplicação (Figura 5).
- *WorkspaceService*: Está classe tem como objetivo lidar com a lógica de negócios relacionada aos *workspaces* (Figura 5). Ela geralmente interage com a API para realizar operações CRUD em *workspaces*.
- *WorkspaceUserService*: Está classe tem como objetivo gerenciar a relação entre usuários e *workspaces*, ela lida com a associação de usuários a diferentes *workspaces*, incluindo permissões e roles (Figura 5).

Figura 6 - Diagrama da página de gerenciamento de workspace

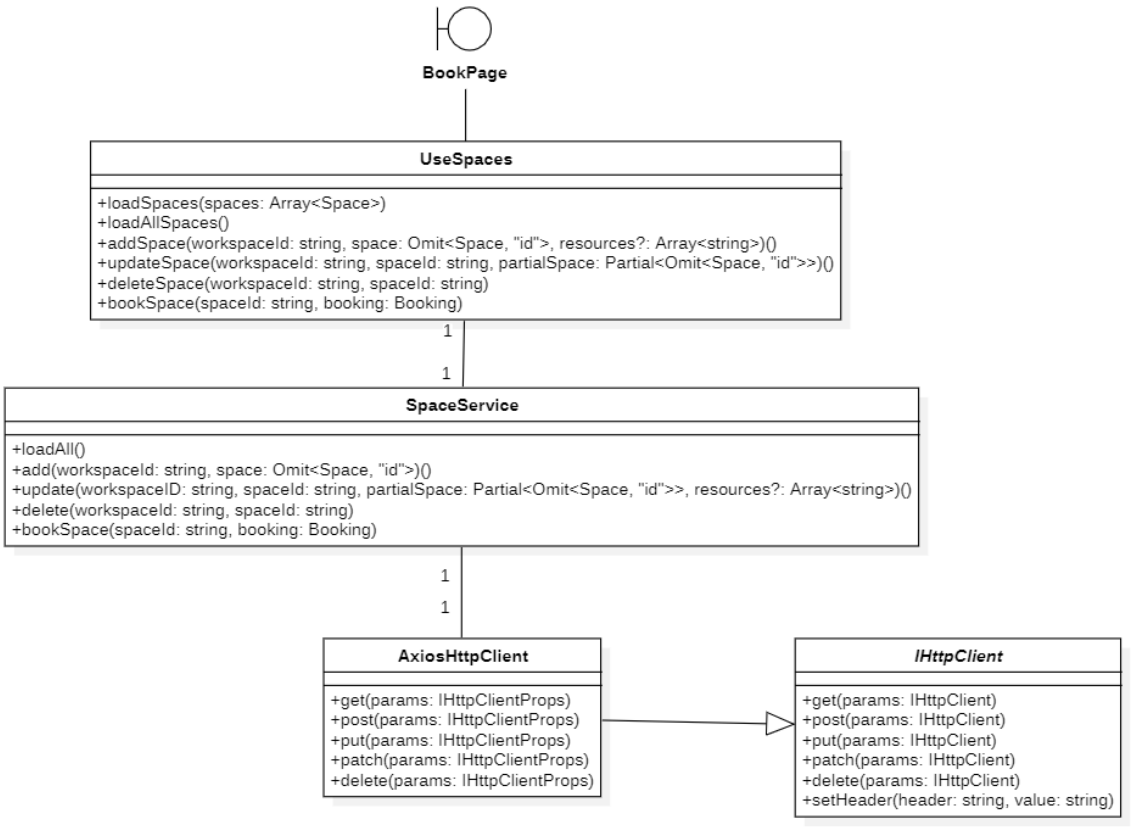


Fonte: Elaborado pelos autores (2025).

A página de gerenciamento de *workspace* (Figura 6), utiliza de oito classes, entretanto *WorkspaceService* (Quadro 32), *WorkspaceUserService* (Quadro 33) e *AxiosHttpClient* (Quadro 30) já foram citados anteriormente, sendo as novas classes são:

- *useSpaces*: Está classe tem como objetivo gerenciar e fornecer o estado e as operações relacionadas aos espaços dentro da aplicação (Figura 6).
- *SpaceService:* Está classe tem como objetivo lidar com a lógica de negócios relacionada aos espaços, ela geralmente interage com API para realizar operações (Figura 6).
- *useWorkspaceDetails:* Está classe tem como objetivo fornecer e gerenciar o estado detalhado de um *workspace* específico, ela é utilizada para acessar e manipular informações detalhadas de um *workspace* (Figura 6).
- *useWorkspaceResource:* Está classe tem como objetivo gerenciar e fornecer o estado e as operações relacionadas aos recursos dentro de um *workspace* específico (Figura 6).
- *WorkspaceResourceService:* Está classe tem como objetivo lidar com a lógica de negócios relacionada aos recursos dentro de *workspaces*, ela interage com a API para realizar operações (Figura 6).

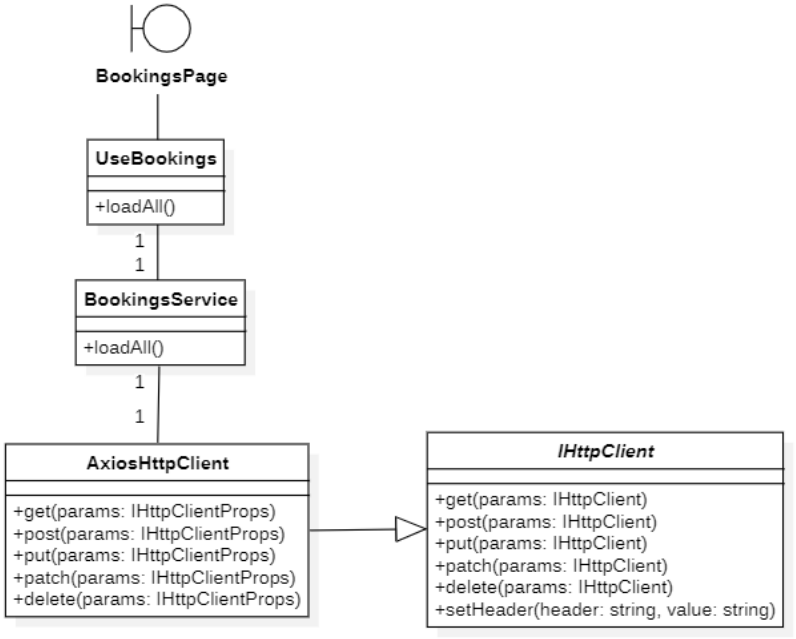
Figura 7 - Diagrama da página de agendamento.



Fonte: Elaborado pelos autores (2025).

A página de agendamento, é onde o usuário reserva um espaço, contendo para isso, três classes, já citadas anteriormente: *useSpace* no Quadro 34, *SpaceService* no Quadro 35 e *AxiosHttpClient* no Quadro 30.

Figura 8 - Diagrama da página de espaços agendados.



Fonte: Elaborado pelos autores (2025).

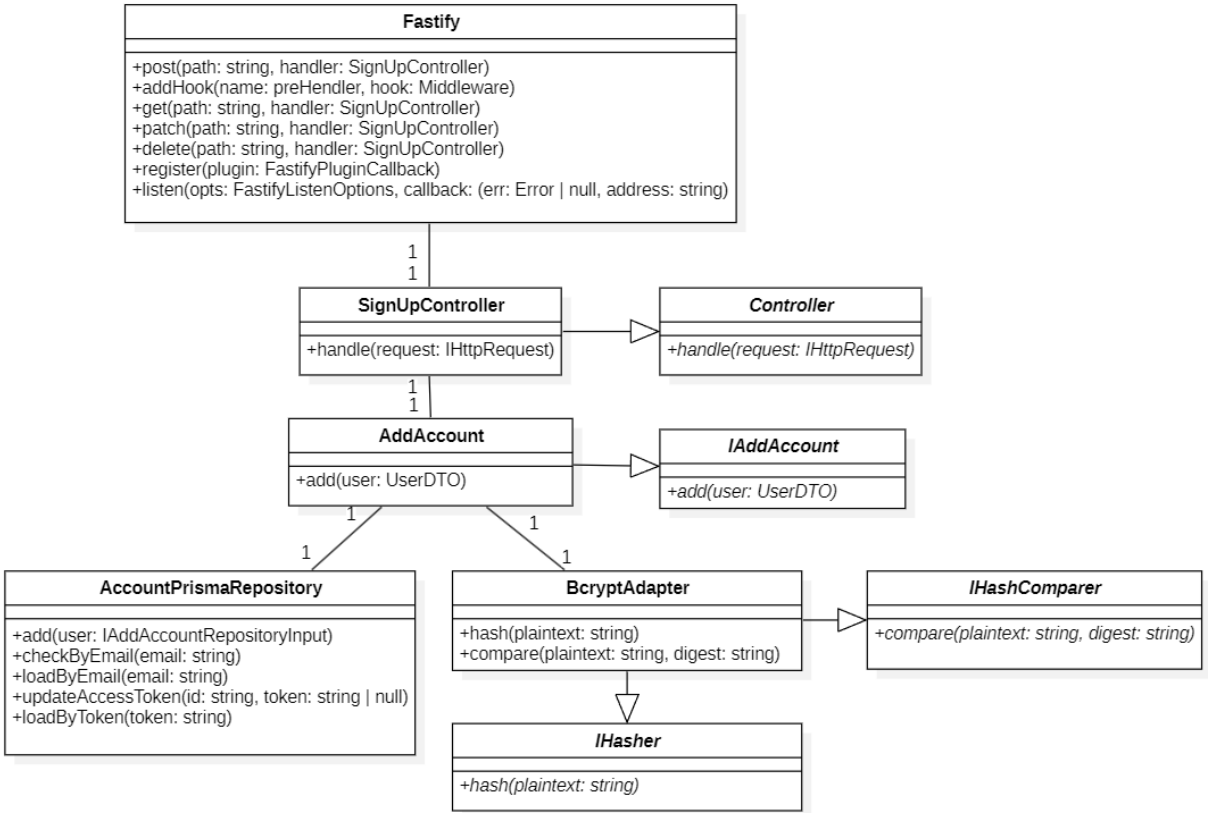
A página de espaços agendamentos, lista os espaços que o usuário reservou (Figura 8). Utilizando o *AxiosHttpClient* já citado anteriormente no Quadro 30 e outras três novas classes para isso:

- *useBookings:* Está classe tem como objetivo gerenciar e fornecer o estado e as operações relacionadas às reservas dentro da aplicação (Figura 8).
- *BookingsService:* Está classe tem como objetivo lidar com a lógica de negócios relacionada às reservas (Figura 8).

3.4 Diagrama de Classe da API

O *back-end* é organizado em 21 rotas, cada uma gerenciada por uma instância do *Fastify*, um framework eficiente para a definição de rotas HTTP. As rotas são protegidas por um middleware de autenticação, exceto para as operações de *signIn* e *signUp*, garantindo a segurança das operações. A arquitetura do *back-end* segue um padrão claro, com cada rota sendo gerenciada por um *controller* específico, que delega a lógica de negócios ao *usecase* correspondente, que utiliza de um repositor para interagir com o banco de dados.

Figura 9 - Diagrama de classe do SignUp.



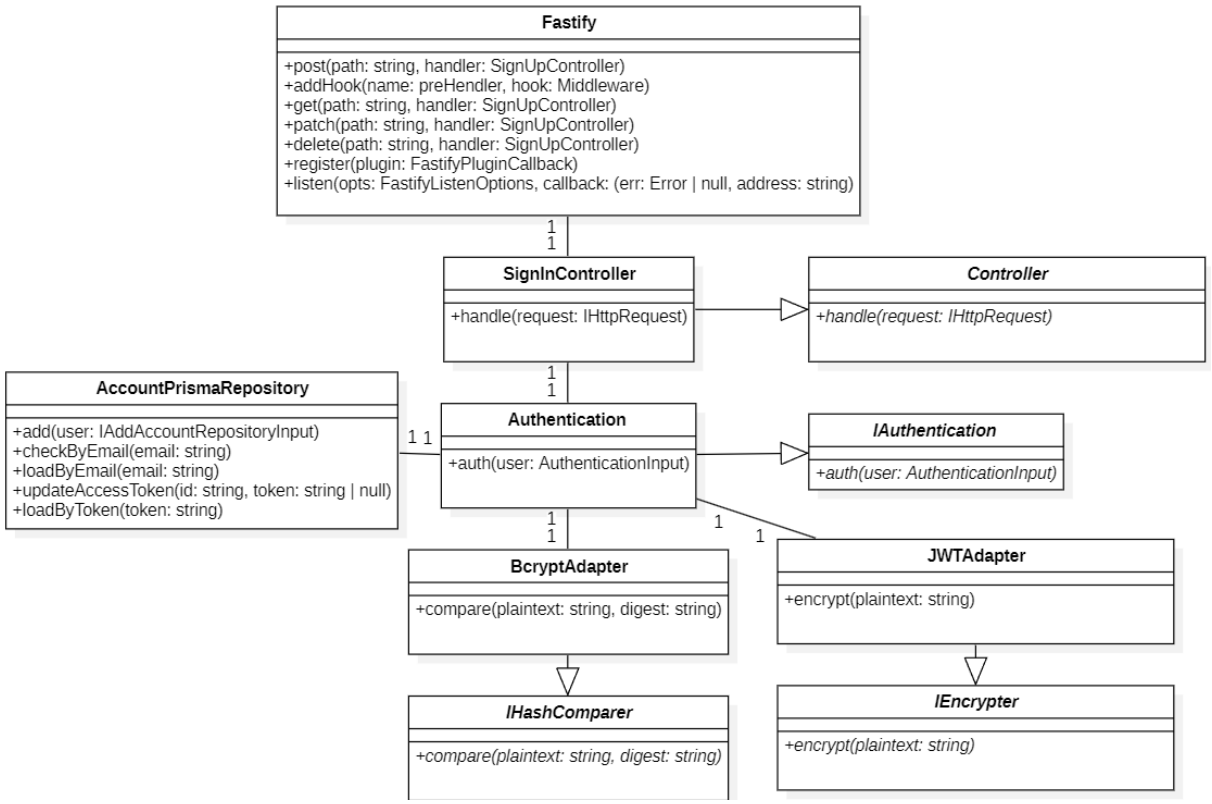
Fonte: Elaborado pelos autores (2025).

Na Figura 9, pode ser vista a rota de cadastro, que possuí cinco classes, sendo elas:

- *Fastify*: É um framework web com uma poderosa arquitetura de plugins (Figura 9).
- *SignUpController*: É uma classe que representa o controlador responsável por lidar com as solicitações de registro de novas contas de usuário em um sistema (Figura 9). Ele implementa a interface *Controller*, que define um método *handle* para lidar com as requisições HTTP.
- *Controller*: é uma interface que contêm os métodos que todo *controller* deve implementar (Figura 9).
- *AddAccount*: Esta classe é responsável por adicionar uma nova conta de usuário ao sistema (Figura 9).
- *IAddAccount*: Esta é uma interface que adiciona uma nova conta de usuário ao sistema (Figura 9).

- *AccountPrismaRepository*: Esta classe é responsável por lidar com operações relacionadas ao armazenamento e recuperação de dados de contas de usuário no banco de dados usando o Prisma ORM (Figura 9).
- *BcryptAdapter*: Esta classe é responsável por encapsular a funcionalidade do pacote *bcrypt* para lidar com a criptografia de senhas usando o algoritmo de *hash Bcrypt* (Figura 9).
- *IHasher*: É a Interface responsável por realizar operações de *hash* em dados, como *strings*, arquivos ou outros tipos de dados (Figura 9).
- *IHashComparer*: É a Interface responsável por realizar a comparação de *hashes* a partir de duas *strings* (Figura 9).

Figura 10 - Diagrama de classe do Sign-In.

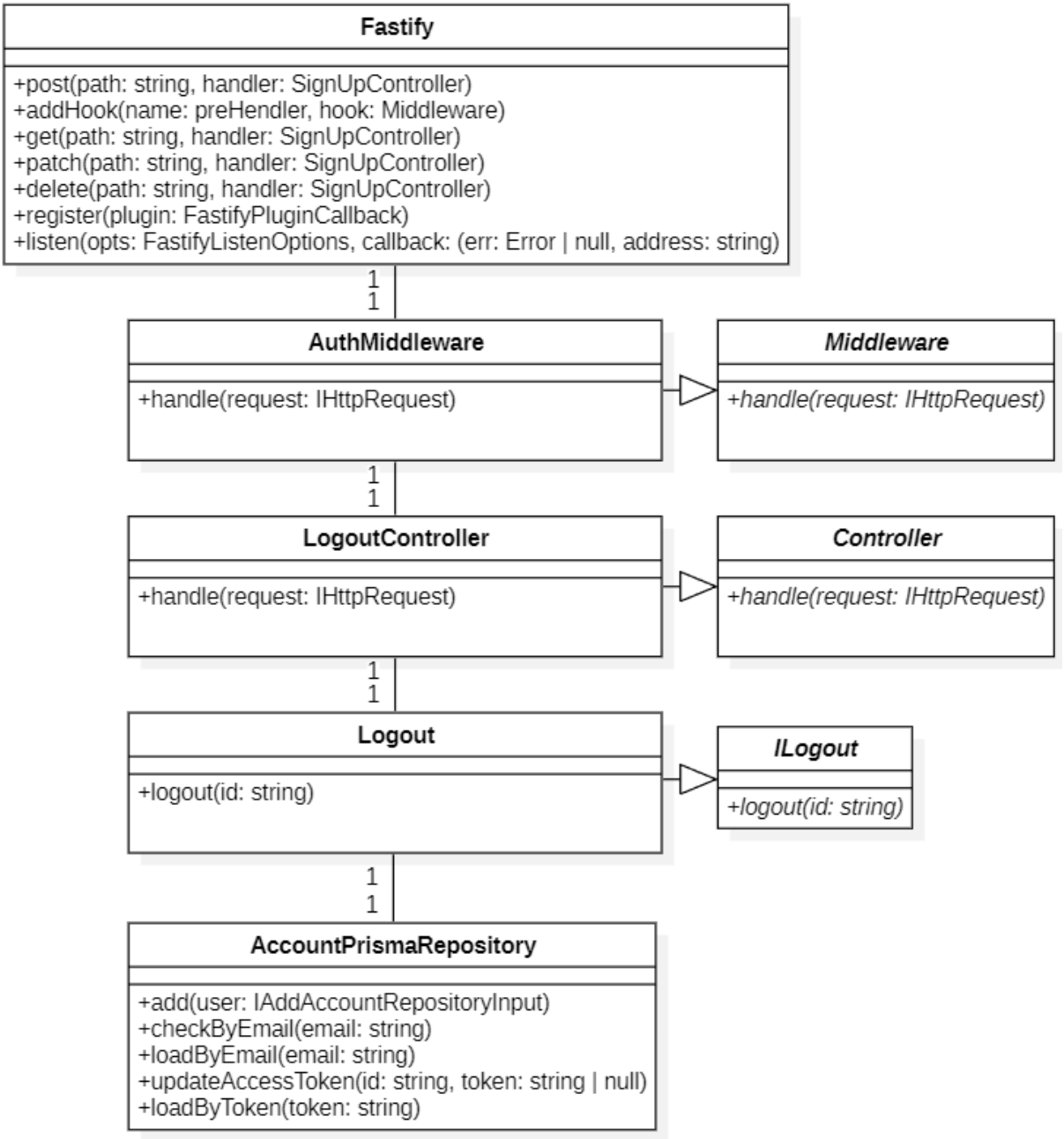


Fonte: Elaborado pelos autores (2025).

A rota de autenticação do usuário (Figura 10) e composta por seis classes e três interfaces. Sendo que, já foi citado: *Fastify* (Quadro 41), *Controller:* (Quadro 43), *AccountPrismaRepository* (Quadro 45), *BcryptAdapter* (Quadro 46), *IHashComparer* (Quadro 48). As classes e interfaces que não foram citadas são:

- *SignInController*: Esta classe é responsável por lidar com as solicitações de autenticação de usuários no sistema, ou seja, o processo de login (Figura 10).
- *Authentication*: Esta classe é responsável por autenticar usuários no sistema, verificando se as credenciais fornecidas são válidas (Figura 10).
- *IAuthentication*: Esta interface é responsável por autenticar usuários no sistema (Figura 10).
- *JWTAdapter*: Esta classe é responsável por encapsular a funcionalidade de criptografia de dados usando *JSON Web Tokens* (Figura 10).
- *IEncrypter*: É a interface responsável por realizar as operações de criptografia (Figura 10).

Figura 11 - Diagrama de classe do Logout.



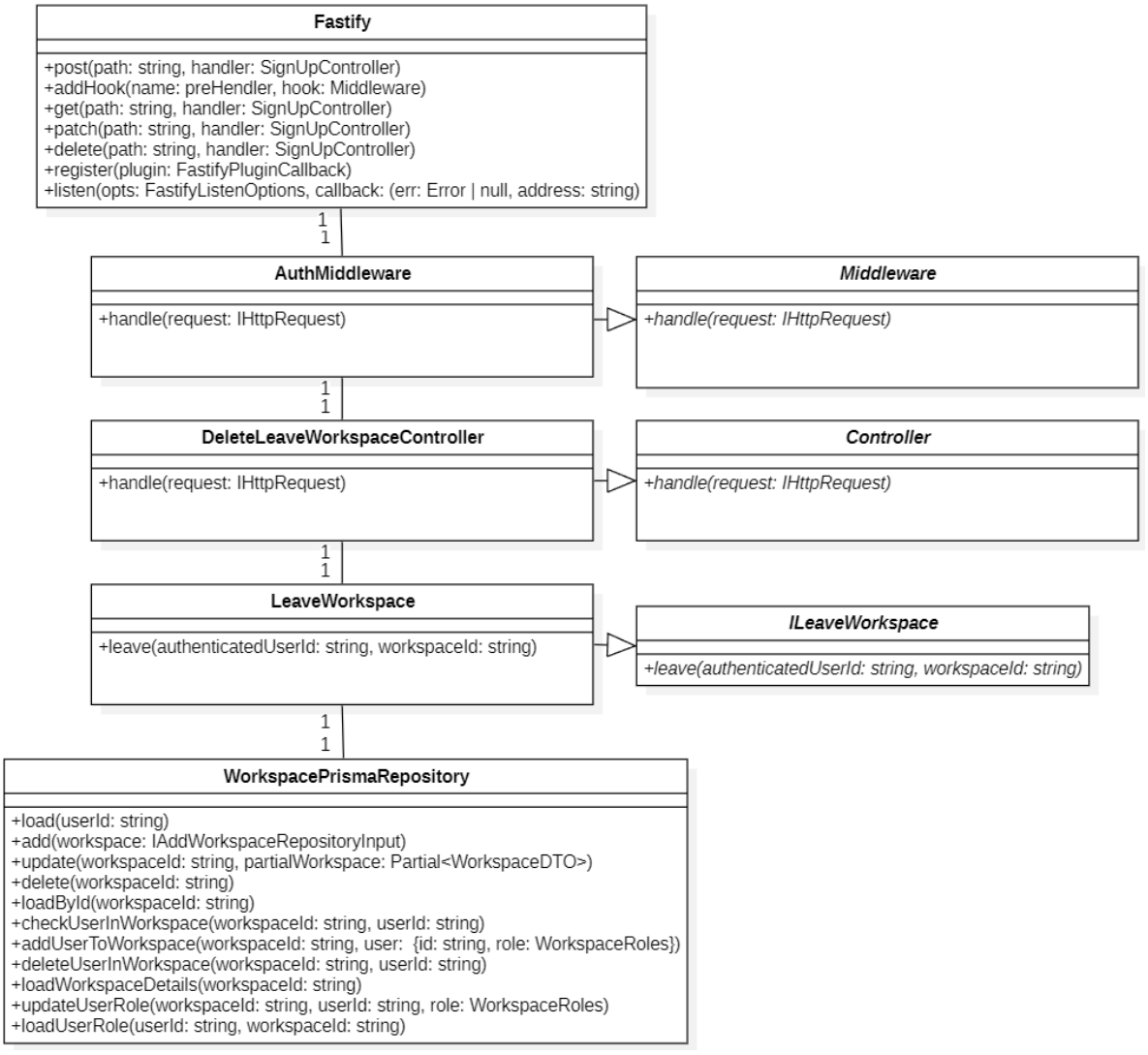
Fonte: Elaborado pelos autores (2025).

A rota de logout (Figura 11), é composta por cinco classes. Sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), e *AccountPrismaRepository* (Quadro 45) já foram citadas. As classes novas são:

- *AuthMiddleware*: Esta classe é responsável por verificar se uma requisição HTTP possui um token de autenticação válido e, se tiver, carregar os dados da conta associada a esse token (Figura 11).

- *Middleware*: Esta é uma interface que define um contrato para um middleware em uma aplicação web. Um middleware é um componente intermediário que processa uma requisição HTTP antes que ela alcance o *endpoint* final (Figura 11).
- *LogoutController*: Este controlador é responsável por lidar com as solicitações de logout de usuários do sistema (Figura 11).
- *Logout*: Esta classe representa o caso de uso de logout de usuários do sistema (Figura 11). Seu propósito é realizar as operações necessárias para finalizar a sessão de um usuário, como invalidar tokens de acesso ou limpar informações de autenticação.
- *ILogout*: Esta interface permite o logout do usuário do sistema (Figura 11).

Figura 12 - Diagrama de classe do DeleteLeave.

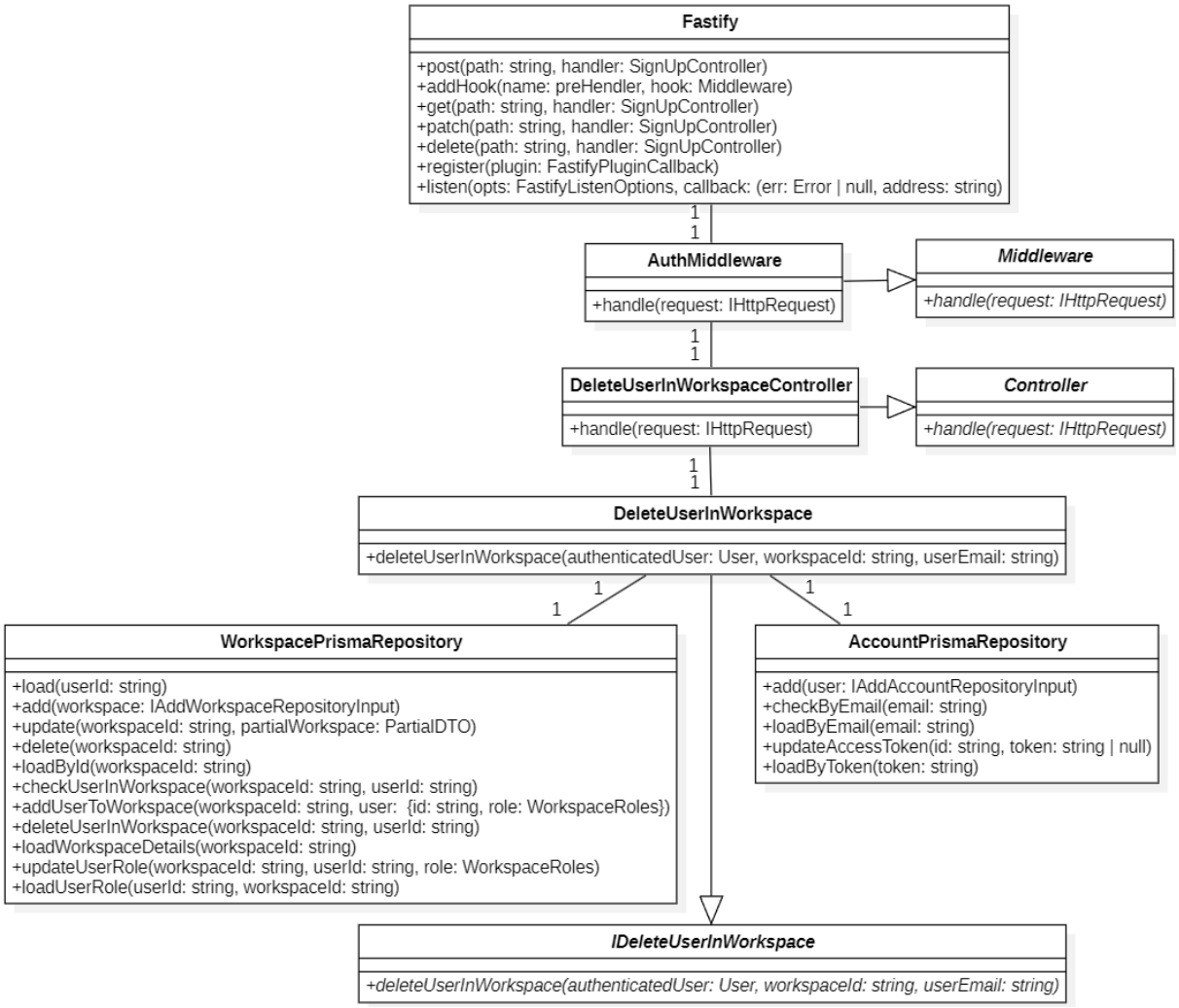


Fonte: Elaborado pelos autores (2025).

A rota de sair da *workspace* (Figura 12), é composta por cinco classes, sendo que, *Fastify (*Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53) e *Middleware* (Quadro 54), já foram citadas, e as novas classes são:

- *DeleteLeaveWorkspaceController*: Este controlador é responsável por lidar com as solicitações de um usuário para sair de um espaço de trabalho (Figura 12).
- *LeaveWorkspace*: Esta classe representa o caso de uso para permitir que um usuário saia de um espaço de trabalho (Figura 12).
- *ILeaveWorkspace*: Esta interface permite que um usuário possa sair de uma workspace (Figura 12).
- *WorkspacePrismaRepository*: O objetivo desta classe é servir como uma camada de abstração entre a aplicação e o banco de dados, especificamente para operações relacionadas a *workspaces* (Figura 12).

Figura 13 - Diagrama de classe do DeleteUser.



Fonte: Elaborado pelos autores (2025).

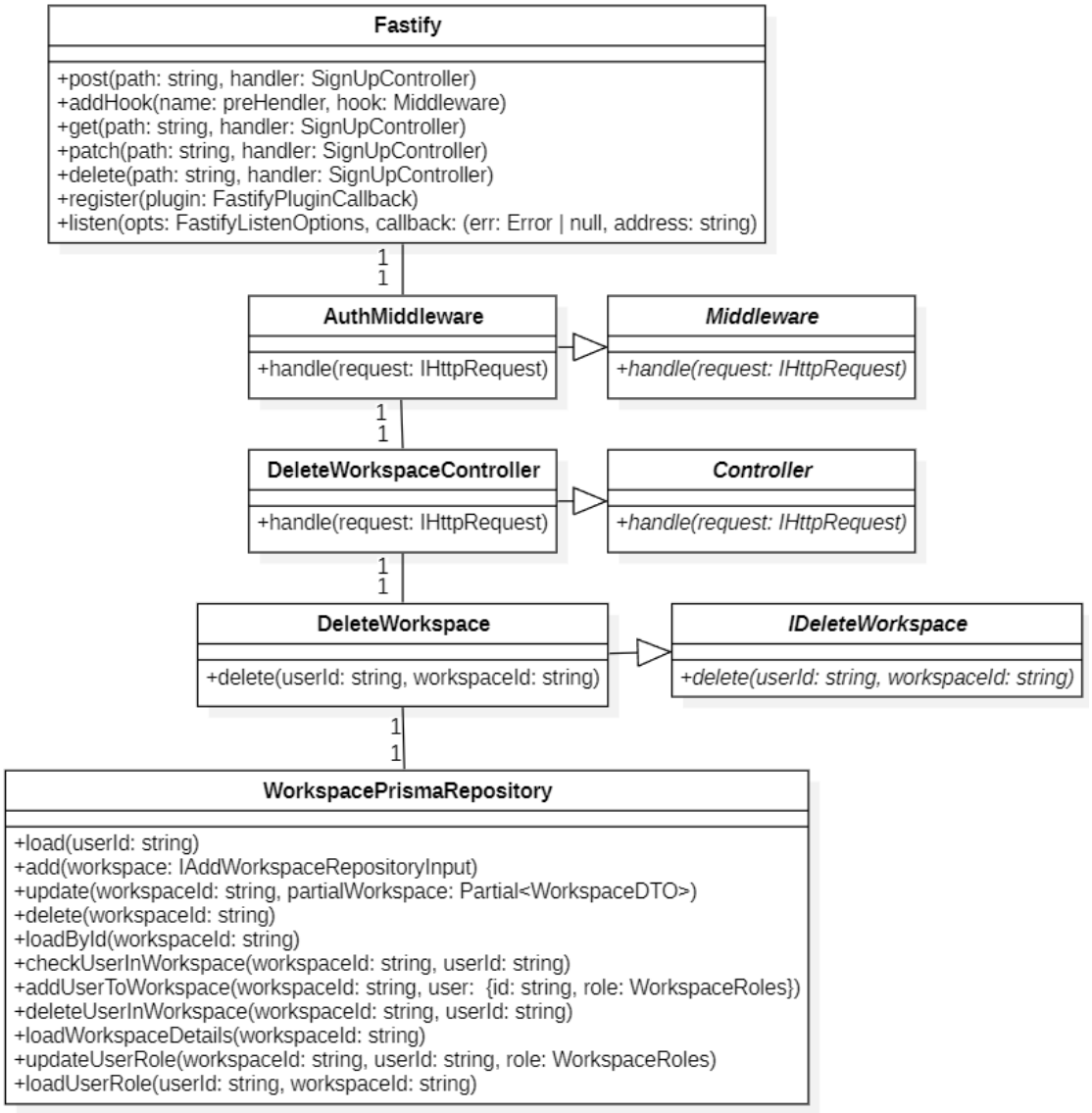
A rota para excluir um usuário (Figura 13), é composta por seis classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AccountPrismaRepository* (Quadro 45), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *WorkspacePrismaRepository* (Quadro 59) já forma citadas. As classes que não foram citadas são:

- *DeleteUserInWorkspaceController*: É o controlador responsável por representar o caso de uso de exclusão de um usuário de um *workspace* (Figura 13). Ela coordena a interação entre os repositórios de usuários e de *workspaces* para realizar a remoção do usuário do *workspace* especificado.
- *DeleteUserInWorkspace*: É a classe responsável por representar o caso de uso de exclusão de um usuário de um *workspace* (Figura 13). Ela coordena

a interação entre os repositórios de usuários, *workspaces* e permissões para realizar a remoção do usuário do *workspace* especificado.

- *DeleteUserInWorkspace*: É a interface responsável por proporcionar a exclusão de um usuário de uma *workspace* (Figura 13).

Figura 14 - Diagrama de classe do DeleteWorkspace.

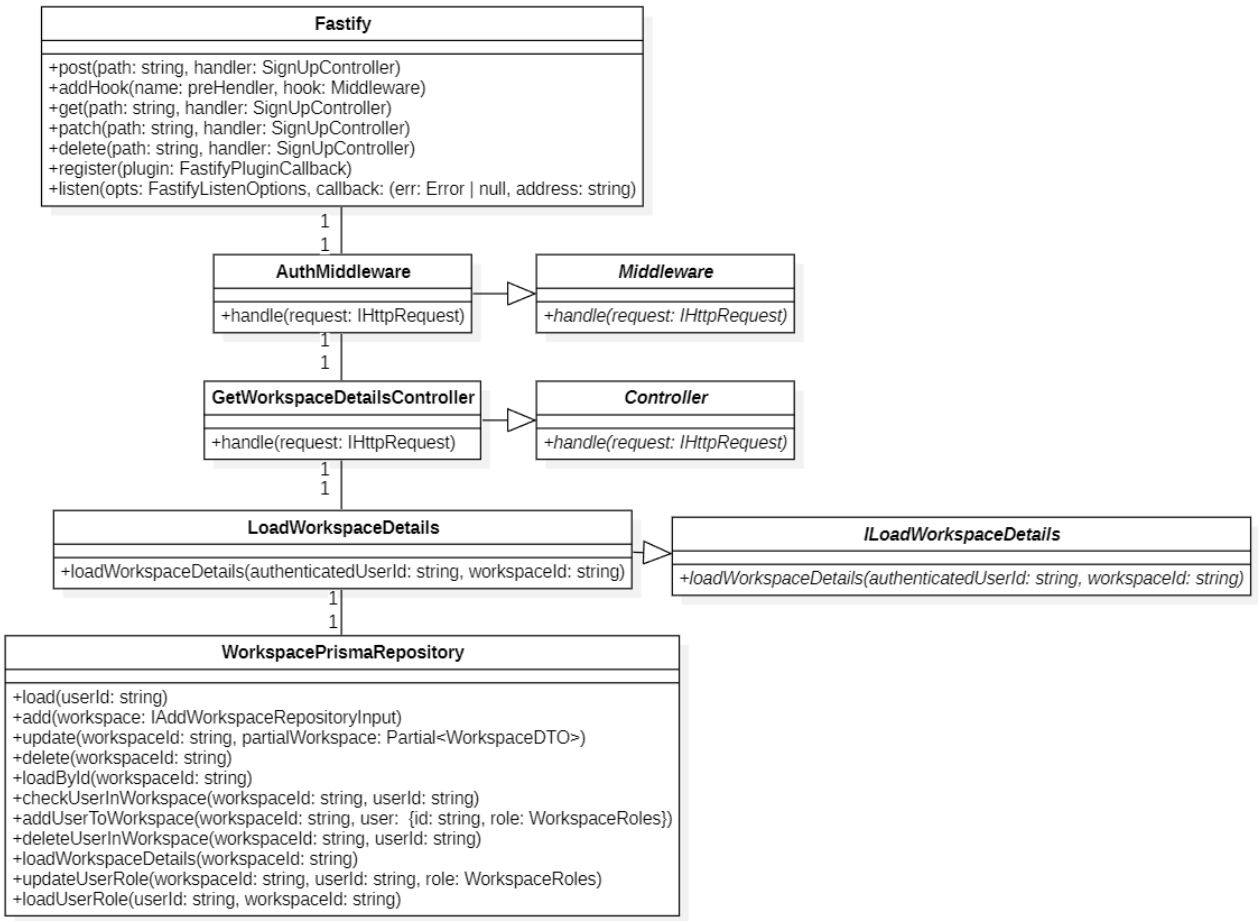


Fonte: Elaborado pelos autores (2025).

A rota para deletar uma *workspace* (Figura 14), é composta por cinco classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *WorkspacePrismaRepository* (Quadro 59), já foram citadas. As novas classes que compõe essa rota para deletar são:

- *DeleteWorkspaceController*: É o controlador responsável por manipular as solicitações de exclusão de um *workspace* (Figura 14). Ele implementa a interface *Controller* e possui um método *handle* que recebe a solicitação HTTP e retorna uma resposta HTTP.
- *DeleteWorkspace*: É a classe responsável por lidar com a exclusão de um workspace (Figura 14). Ela recebe a carga útil da solicitação contendo o ID do usuário autenticado e o ID do *workspace* a ser excluído. A classe utiliza o repositório *ILoadUserRoleRepository* para carregar o papel do usuário autenticado no workspace e o repositório *IDeleteWorkspaceRepository* para efetuar a exclusão do *workspace*.
- *IDeleteWorkspace*: É a interface responsável por cuidar a exclusão de uma *workspace* (Figura 14).

Figura 15 - Diagrama de classe do GetWorkspaceDetails.

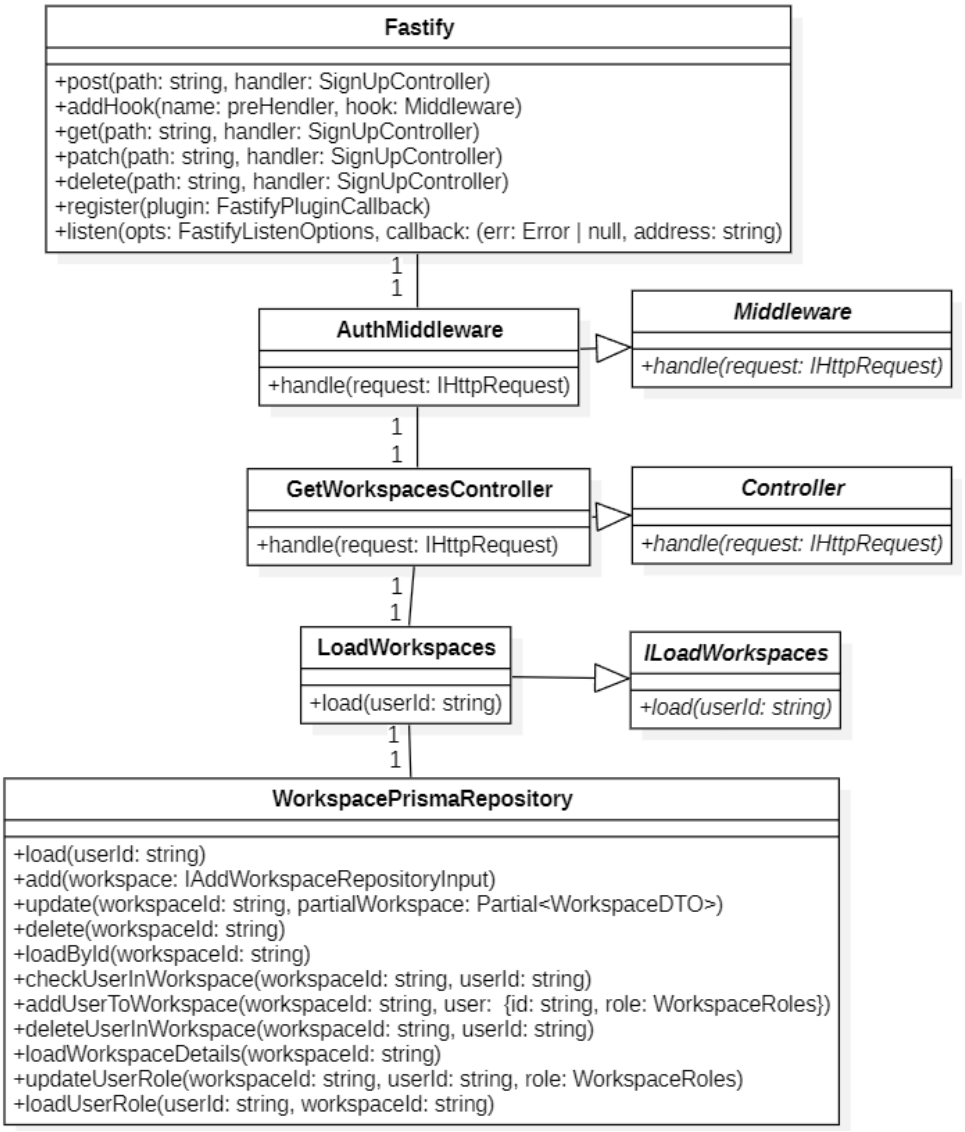


Fonte: Elaborado pelos autores (2025).

A rota para obter detalhes da *workspace* (Figura 15) é composta por cinco classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *WorkspacePrismaRepository* (Quadro 59), já foram citadas. As classes que não foram citadas são as seguintes:

- *GetWorkspaceDetailsController*: É o controlador responsável por lidar com as solicitações de detalhes do espaço de trabalho (Figura 15). Ele recebe uma instância de *LoadWorkspaceDetails* no construtor para buscar os detalhes do espaço de trabalho.
- *LoadWorkspaceDetails*: É a classe responsável por carregar os detalhes de um espaço de trabalho, incluindo informações como nome, descrição, membros e permissões associadas (Figura 15). Ela recebe instâncias de repositórios no construtor para realizar operações relacionadas aos detalhes do espaço de trabalho.
- *ILoadWorkspaceDetails*: É a interface responsável por carregar as informações de uma *workspace* (Figura 15).

Figura 16 - Diagrama de classe do GetWorkspace.



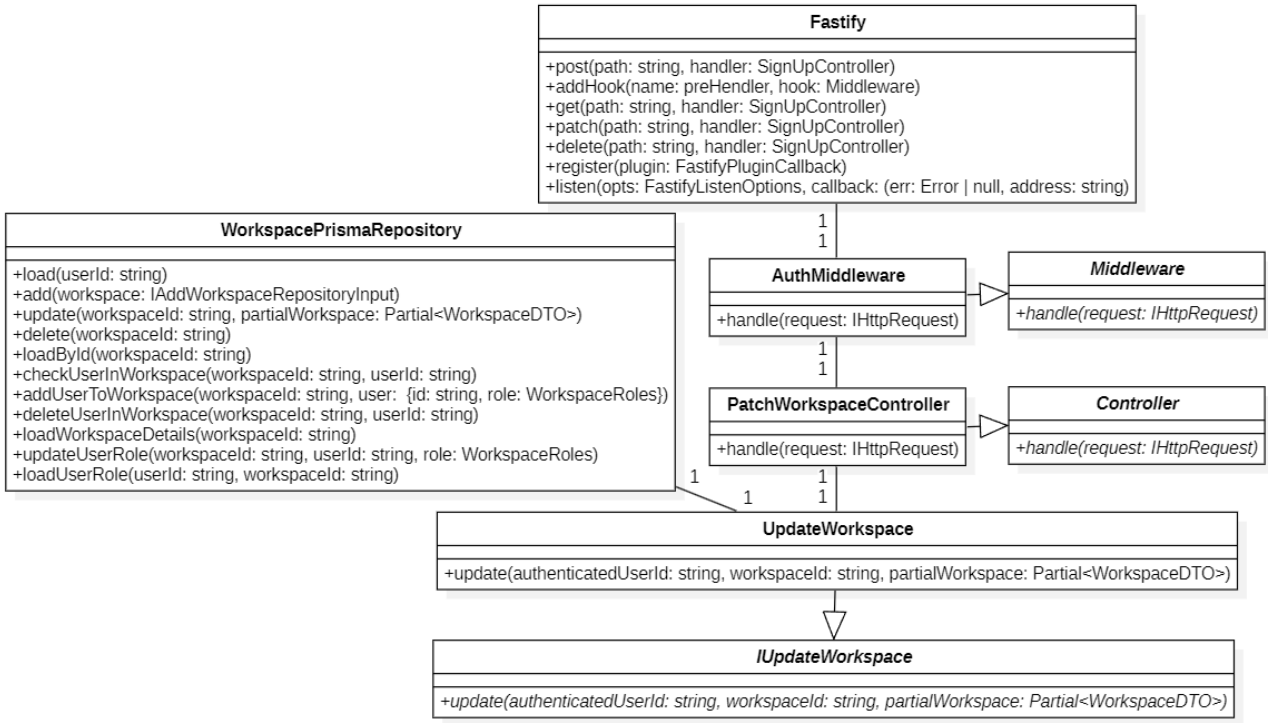
Fonte: Elaborado pelos autores (2025).

A rota para obter a lista de *workspace* (Figura 16) é composta por cinco classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *WorkspacePrismaRepository* (Quadro 59), já foram citadas. As demais classes que não foram citadas ainda são:

- *GetWorkspaceController*: É responsável por lidar com as requisições HTTP para obter os espaços de trabalho de um usuário (Figura 16). Ele implementa a interface *Controller* e tem um método *handle* que recebe uma requisição HTTP e retorna uma resposta HTTP.
- *LoadWorkspace*: É uma classe responsável por carregar os espaços de trabalho de um usuário (Figura 16).

- *ILoadWorkspace*: É a interface responsável por carregar as *workspaces* relacionadas a um determinado usuário (Figura 16).

Figura 17 - Diagrama de classe do PatchWorkspace

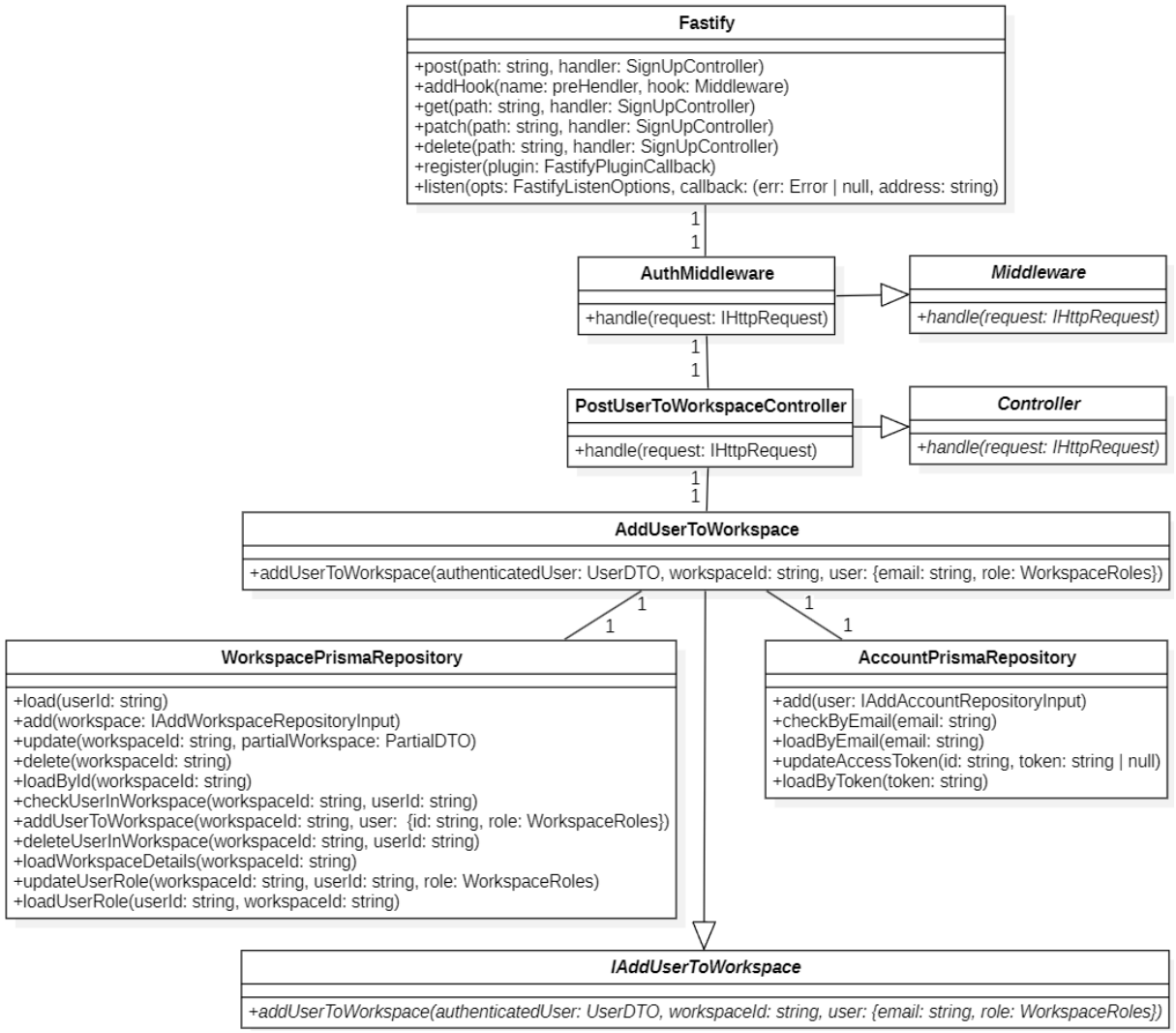


Fonte: Elaborado pelos autores (2025).

A rota para atualizar uma *workspace* (Figura 17) é composta por cinco classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *WorkspacePrismaRepository* (Quadro 59), já foram citadas. As classes que não foram citadas para a atualização de uma *workspace* são:

- *PatchWorkspaceController*: É o controlador responsável por lidar com requisições PATCH para atualizar informações de um espaço de trabalho (Figura 17).
- *UpdateWorkspace*: É a classe responsável por atualizar informações de um espaço de trabalho (Figura 17).
- *IUpdateWorkspace*: É a interface responsável por atualizar as informações de uma *workspace* (Figura 17).

Figura 18 - Diagrama de classe do PostUserToWorkspace



Fonte: Elaborado pelos autores (2025).

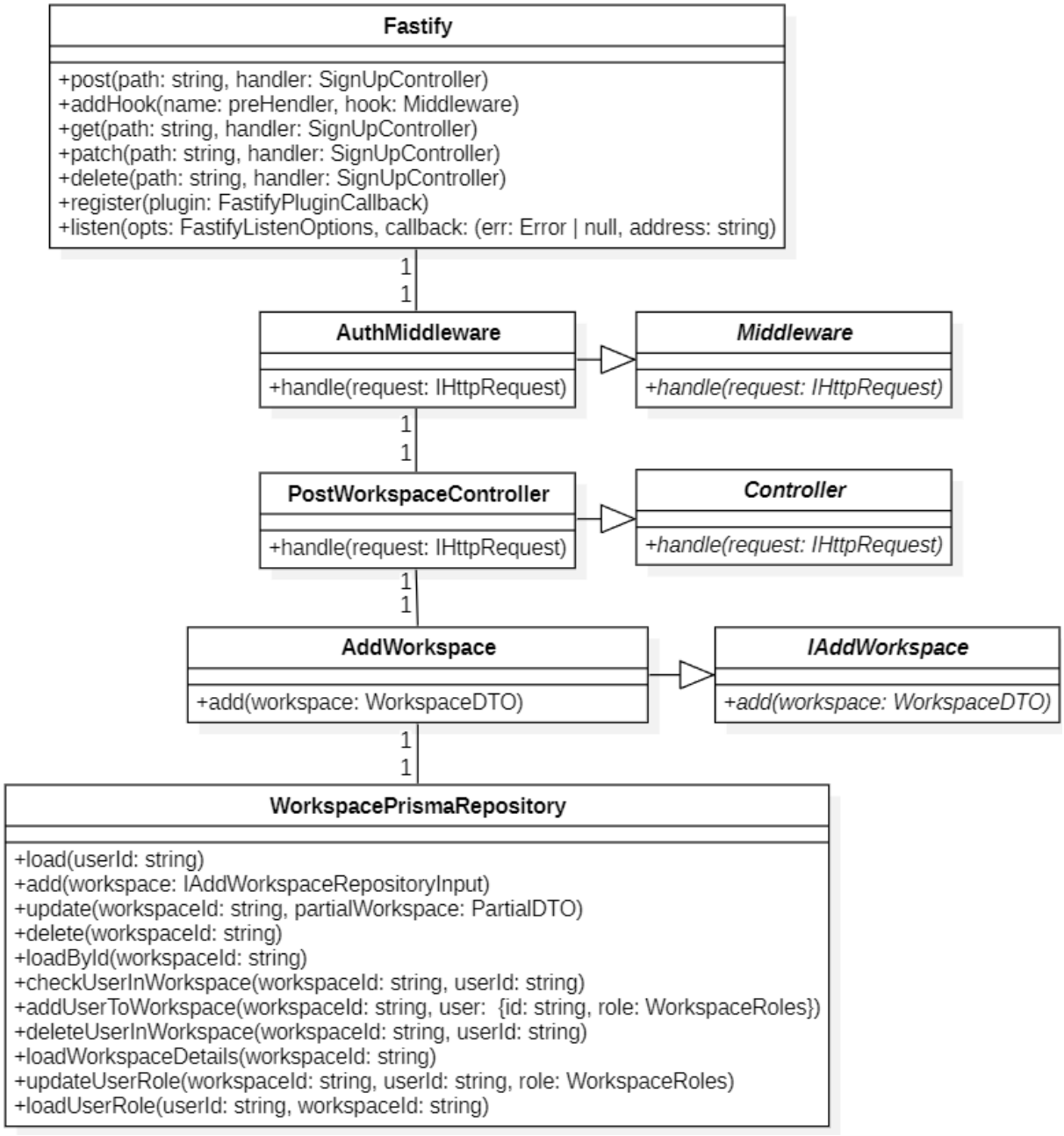
A rota para adicionar um usuário a uma *workspace* (Figura 18) é composta por seis classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54), *AccountPrismaRepository* (Quadro 45) e *WorkspacePrismaRepository* (Quadro 59), já foram citadas. As classes que não foram citadas anteriormente são:

- *PostUserToWorkspaceController*: É o controlador responsável por lidar com a adição de um usuário a um espaço de trabalho (Figura 18).
- *AddUserToWorkspace*: Seu Objetivo é encapsular a lógica relacionada à adição de um usuário a um espaço de trabalho (Figura 18). Ela coordena as interações entre os diversos componentes do sistema, como os repositórios

de dados e a lógica de negócios, garantindo que a operação seja executada de acordo com as regras e permissões estabelecidas.

- *IAddUserToWorkpsace*: É a interface responsável por adicionar um usuário a uma *workspace* (Figura 18).

Figura 19 - Diagrama de classe do PostWorkspace

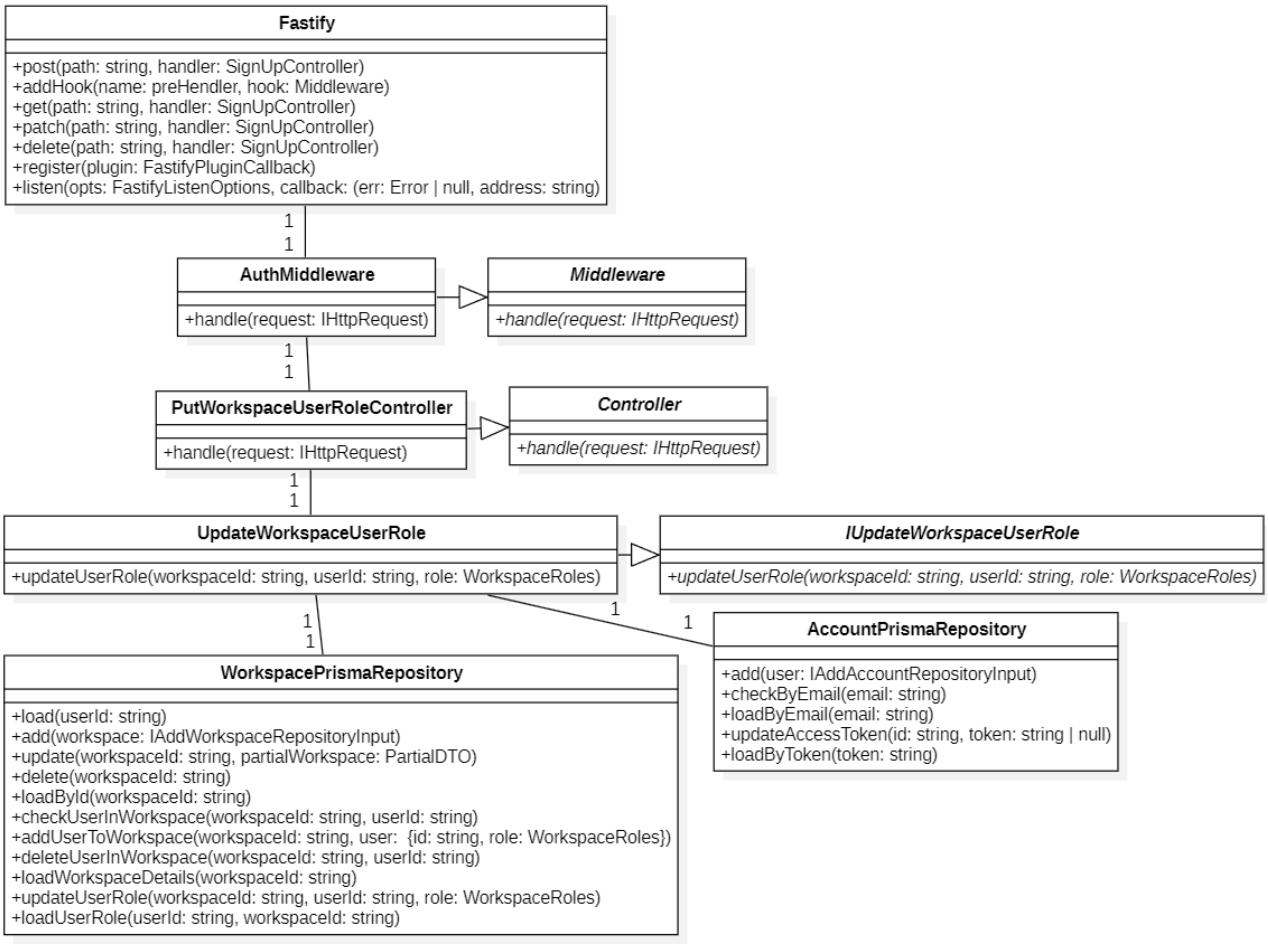


Fonte: Elaborado pelos autores (2025).

A rota para à criação de um novo espaço de trabalho a uma *workspace* (Figura 19) é composta por cinco classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *WorkspacePrismaRepository* (Quadro 59), já foram citadas. As classes que não foram citadas anteriormente são:

- *PostWorkspaceController*: É responsável por lidar com as solicitações HTTP relacionadas à criação de um novo espaço de trabalho (Figura 19). Ela implementa a interface *Controller* e contém o método *handle*, que processa a requisição recebida e retorna uma resposta adequada.
- *AddWorkspace*: É a classe responsável por definir a lógica de negócio relacionada à adição de um novo espaço de trabalho (Figura 19). Ela recebe como entrada um objeto do tipo *workspace*, que representa as informações necessárias para criar um espaço de trabalho, exceto pelo ID, que geralmente é gerado automaticamente pelo sistema.
- *IAddWorkpsace*: É a interface responsável por realizar as operações necessárias para adicionar um espaço a uma *workspace* (Figura 19).

Figura 20 - Diagrama de classe do PutUserRoleInWorkspace

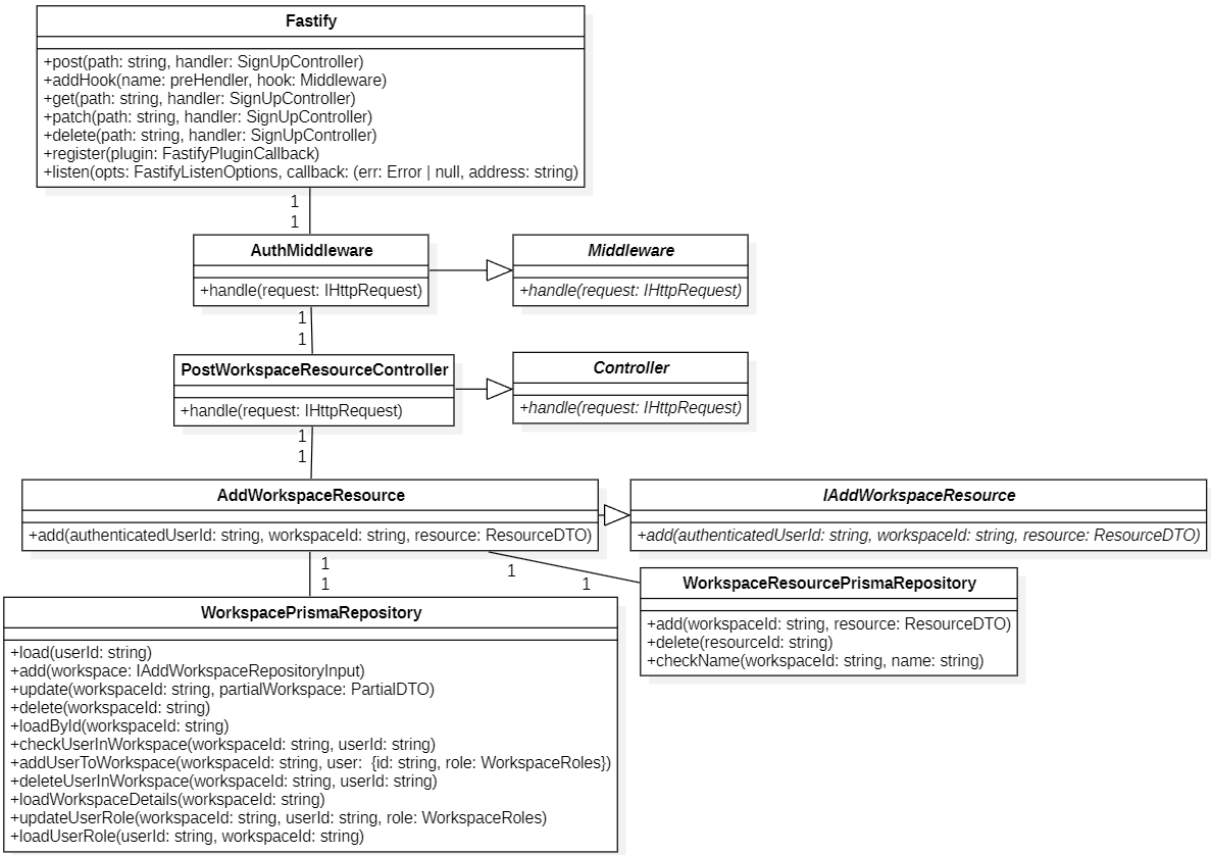


Fonte: Elaborado pelos autores (2025).

A rota para processar as definição ou alteração das funções ou cargos dos usuários (Figura 20) é composta por seis classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54), *AccountPrismaRepository* (Quadro 45) e *WorkspacePrismaRepository* (Quadro 59), já foram citadas. As classes que fazem parte e não foram citadas anteriormente são:

- *PostWorkspaceUserRoleController*: É o controlador responsável por processar as definição ou alteração das funções ou cargos dos usuários na *workspace* (Figura 20).
- *UpdateWorkspaceUserRole*: É a classe responsável por alterar as funções ou cargos de determinados usuários em uma *workspace* (Figura 20).
- IUpdateWorkspaceUserRole: É a interface responsável por realizar a alteração de funções ou cargos de usuários em determinada *workspace* (Figura 20).

Figura 21 - Diagrama de classe do PostWorkspaceResource

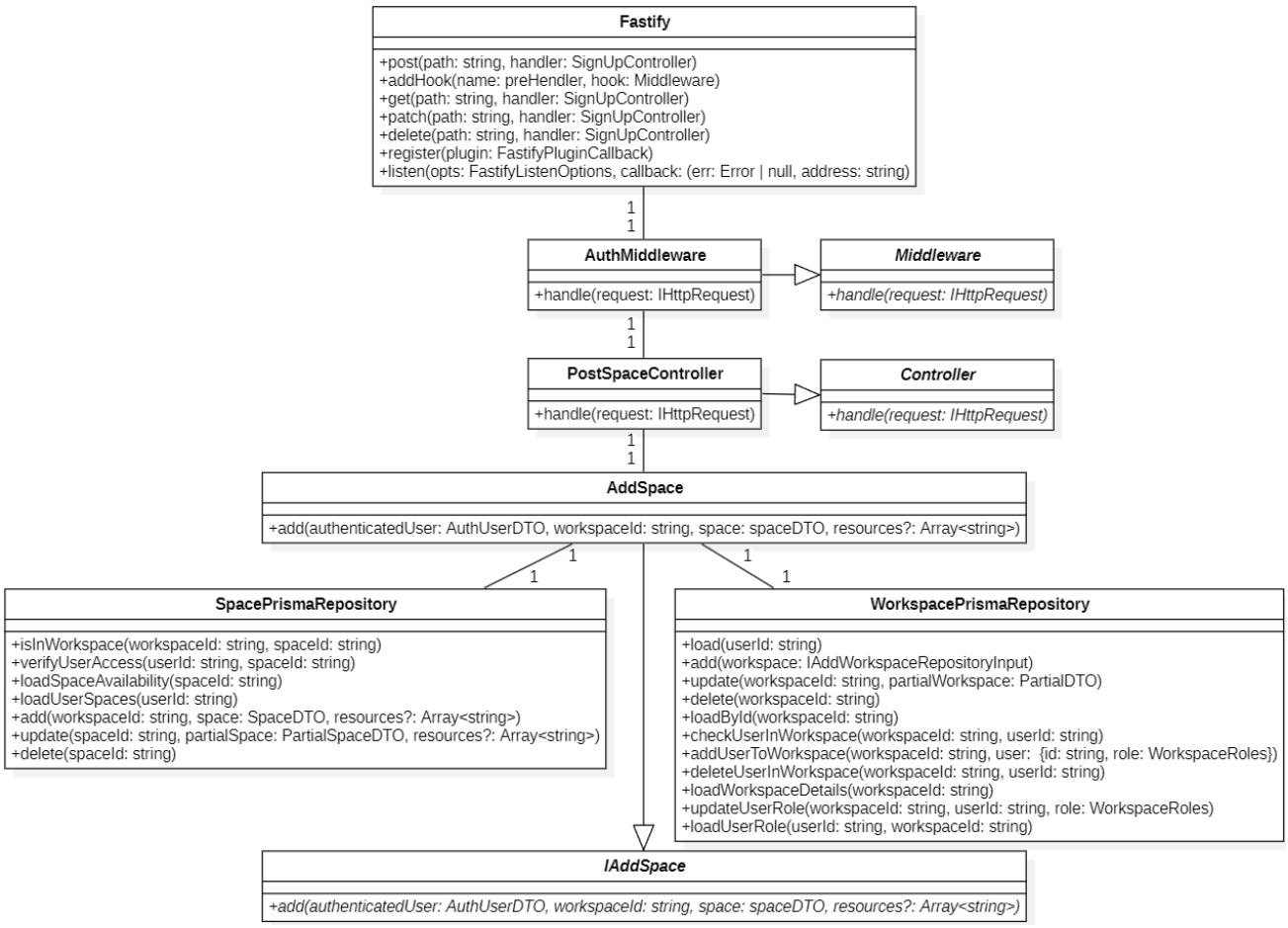


Fonte: Elaborado pelos autores (2025).

A rota que gerencia a adição de recursos ao *workspace* (Figura 21) é composta por seis classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *WorkspacePrismaRepository* (Quadro 59), já foram citadas. As classes que não foram citadas anteriormente e compõem esta rota respectivamente são:

- *PostWorkspaceResourceController*: Um controlador que gerencia a adição de recursos ao *workspace* (Figura 21).
- *AddWorkspaceResource*: É a classe responsável por adicionar recursos ao *workspace* (Figura 21).
- *IAddWorkspaceResource*: É a Interface que define o contrato para adicionar recursos ao *workspace* (Figura 21).
- *WorkspaceResourcePrismaRepository:* é a classe de Repositório que gerencia recursos do *workspace* utilizando Prisma (Figura 21).

Figura 22 - Diagrama de classe do PostSpace

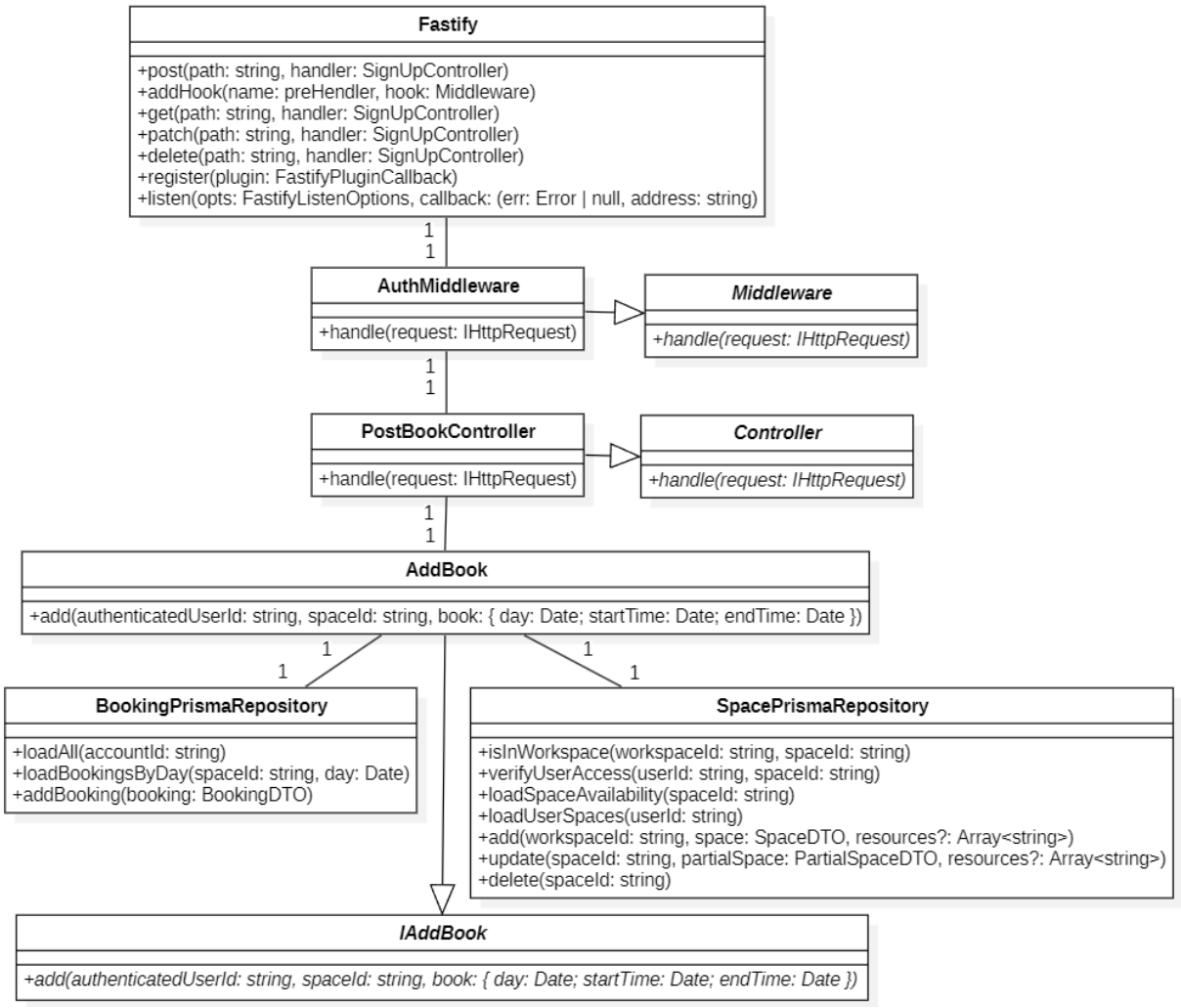


Fonte: Elaborado pelos autores (2025).

A rota que gerencia a criação de novos espaços (Figura 22) é composta por seis classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *WorkspacePrismaRepository* (Quadro 59), já foram citadas. As classes que não foram citadas anteriormente são:

- *PostSpaceController*: Um controlador que gerencia a criação de novos espaços (Figura 22).
- *AddSpace*: É a classe responsável por adicionar novos espaços (Figura 22).
- *IAddSpace*: É a interface responsável por definir o contrato para adicionar novos espaços (Figura 22).
- *SpacePrismaRepository:* é a classe de repositório que gerencia espaços utilizando Prisma (Figura 22).

Figura 23 - Diagrama de classe do PostBook



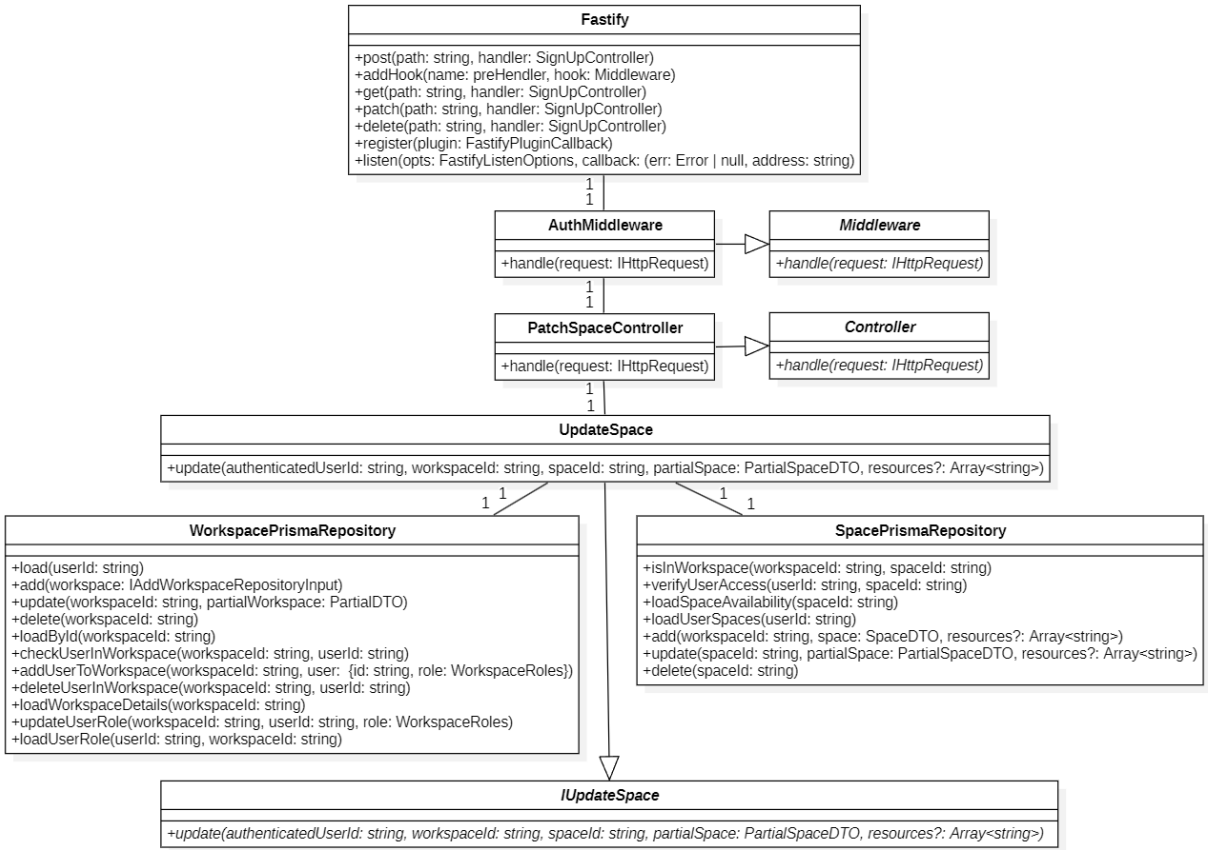
Fonte: Elaborado pelos autores (2025).

A rota que gerencia a requisições de reserva (Figura 23) é composta por seis classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *SpacePrismaRepository* (Quadro 81), já foram citadas. As classes que fazem parte desta rota em questão e não foram citadas anteriormente são:

- *PostBookController*: Um controlador específico para lidar com requisições de reserva (Figura 23).
- *AddBook*: É a classe responsável por implementar a interface *IAddBook* e lida com a lógica de adicionar reservas (Figura 23).
- *AddBook*: É uma interface que define o método para adicionar uma reserva (Figura 23).

- *BookingPrismaRepository*: é uma classe que lida com operações relacionadas a reservas no banco de dados (Figura 23).

Figura 24 - Diagrama de classe do PatchSpace

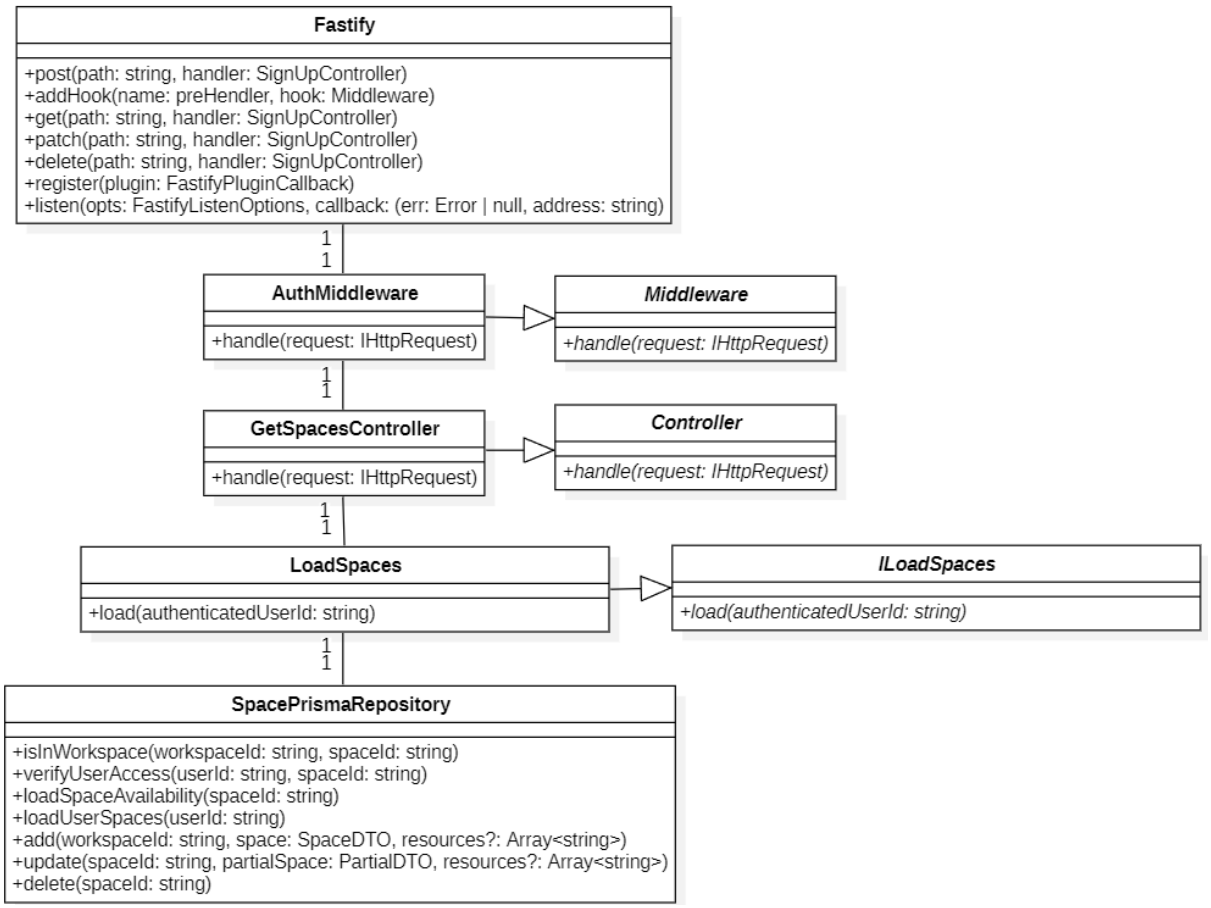


Fonte: Elaborado pelos autores (2025).

A rota para lidar com atualizações de espaços (Figura 24) é composta por seis classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54), *SpacePrismaRepository* (Quadro 81) e *WorkspacePrismaRepository* (Quadro 59), já foram citadas. As classes que não foram citadas anteriormente são:

- *PatchSpaceController*: é o controlador específico para lidar com atualizações de espaços (Figura 24).
- *UpdateSpace*: É uma classe que implementa a interface *IUpdateSpace* e lida com a lógica de atualização de espaços (Figura 24).
- *IUpdateSpace*: É uma interface que define o método para atualizar um espaço (Figura 24).

Figura 25 - Diagrama de classe do GetSpaces

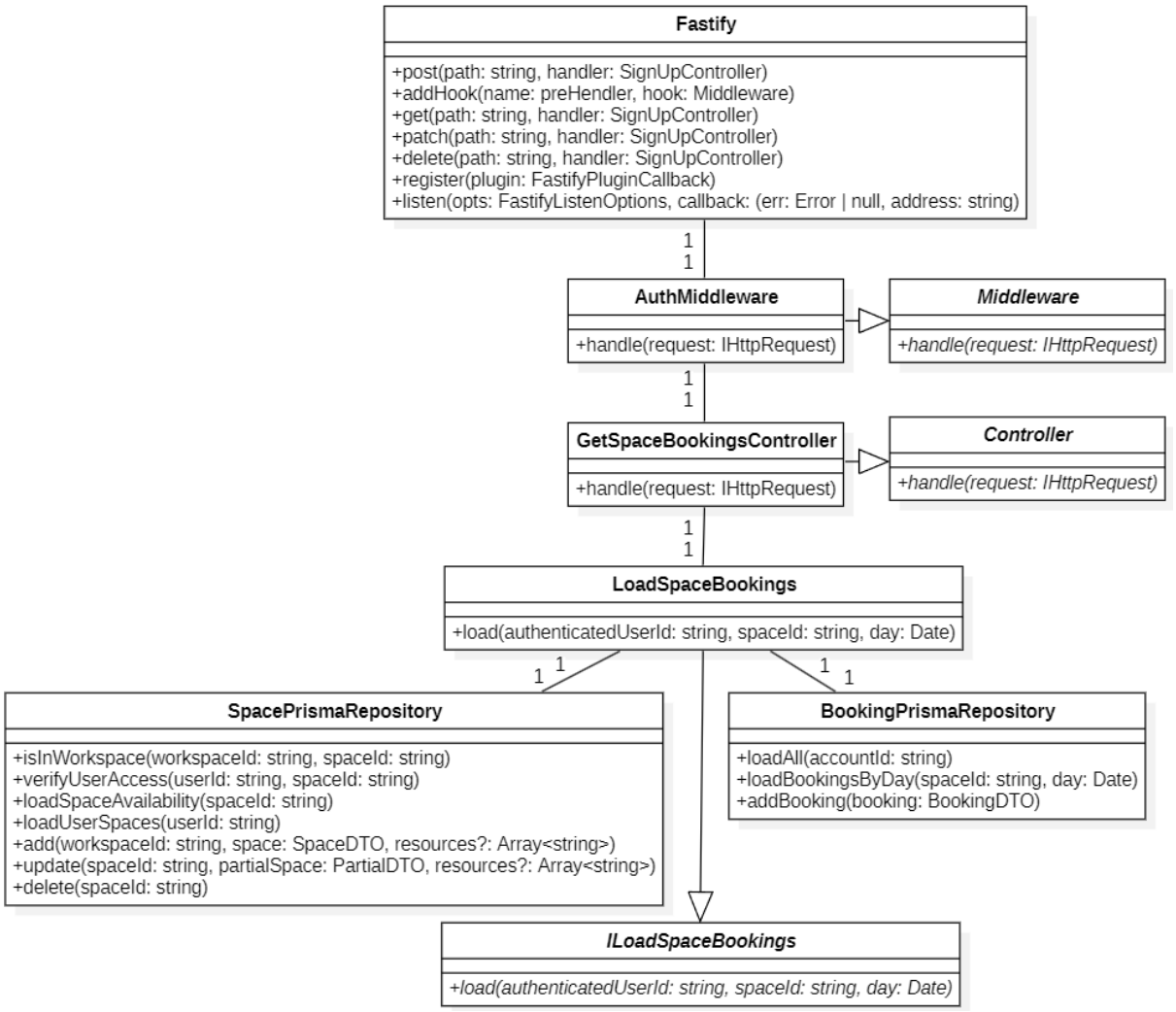


Fonte: Elaborado pelos autores (2025).

A rota para lidar a obtenção de espaços (Figura 25) é composta por cinco classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *SpacePrismaRepository* (Quadro 81), já foram citadas. As demais classes que não foram citadas anteriormente são:

- *GetSpacesController*: É o controlador específico para lidar com a obtenção de espaços (Figura 25).
- *LoadSpaces*: É uma classe que implementa a interface *ILoadSpaces* e lida com a lógica de carregamento de espaços (Figura 25).
- *ILoadSpaces*: É uma interface que define o método para carregar espaços (Figura 25).

Figura 26 - Diagrama de classe do GetSpaceBookings

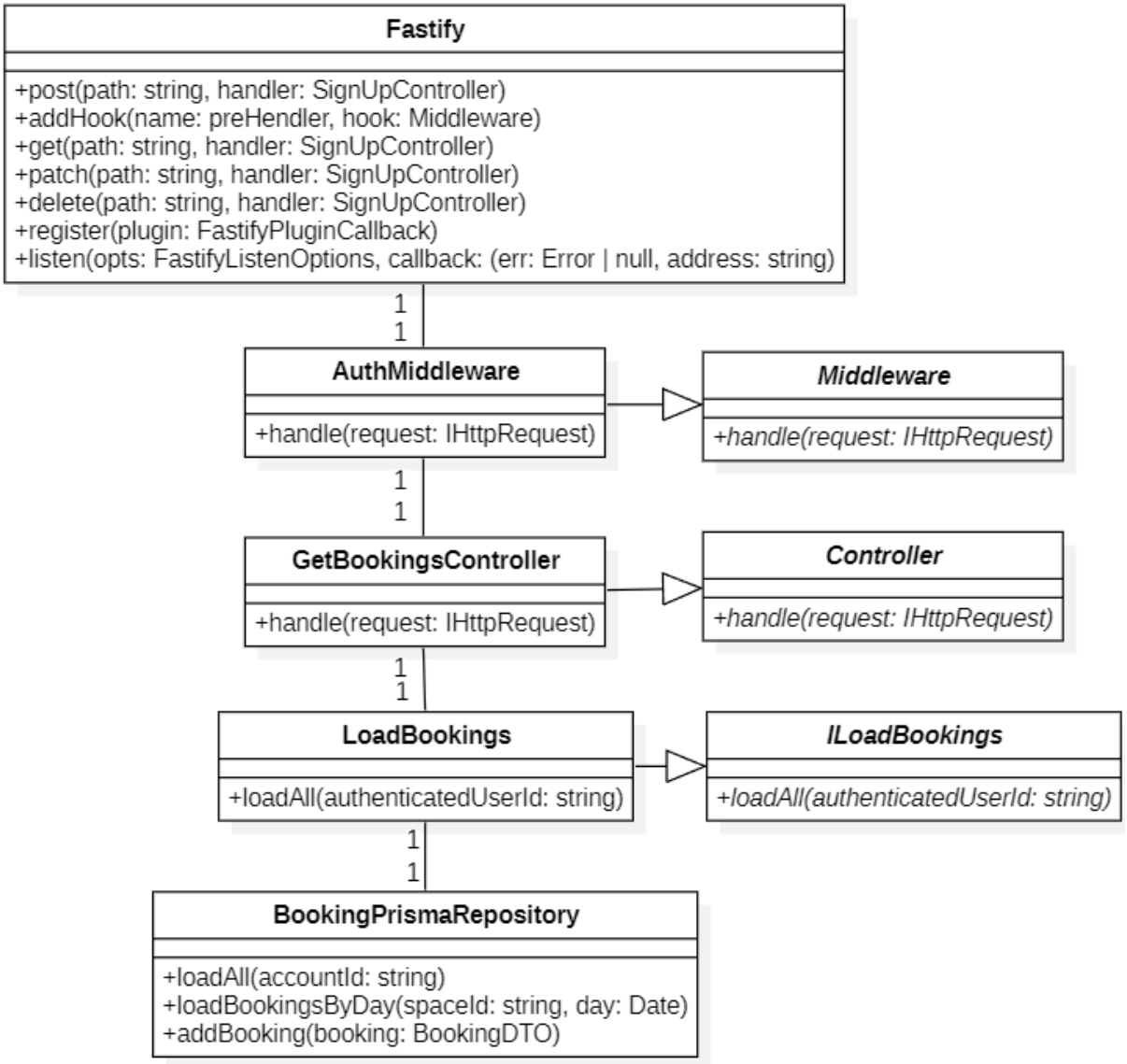


Fonte: Elaborado pelos autores (2025).

A rota para lidar com a obtenção de reservas de espaço (Figura 26) é composta por seis classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54), *SpacePrismaRepository* (Quadro 81) e *BookingPrismaRepository* (Quadro 84), já foram citadas. As classes que não foram citadas anteriormente e fazem parte desta rota em questão são:

- *GetSpaceBookingsController*: É o Controlador específico para lidar com a obtenção de reservas de espaço (Figura 26).
- *LoadSpaceBookings*: É a classe responsável por carregar as reservas de espaço (Figura 26).
- *ILoadSpaceBookings*: É a Interface para carregamento de reservas de espaço (Figura 26).

Figura 27 - Diagrama de classe do GetBookings

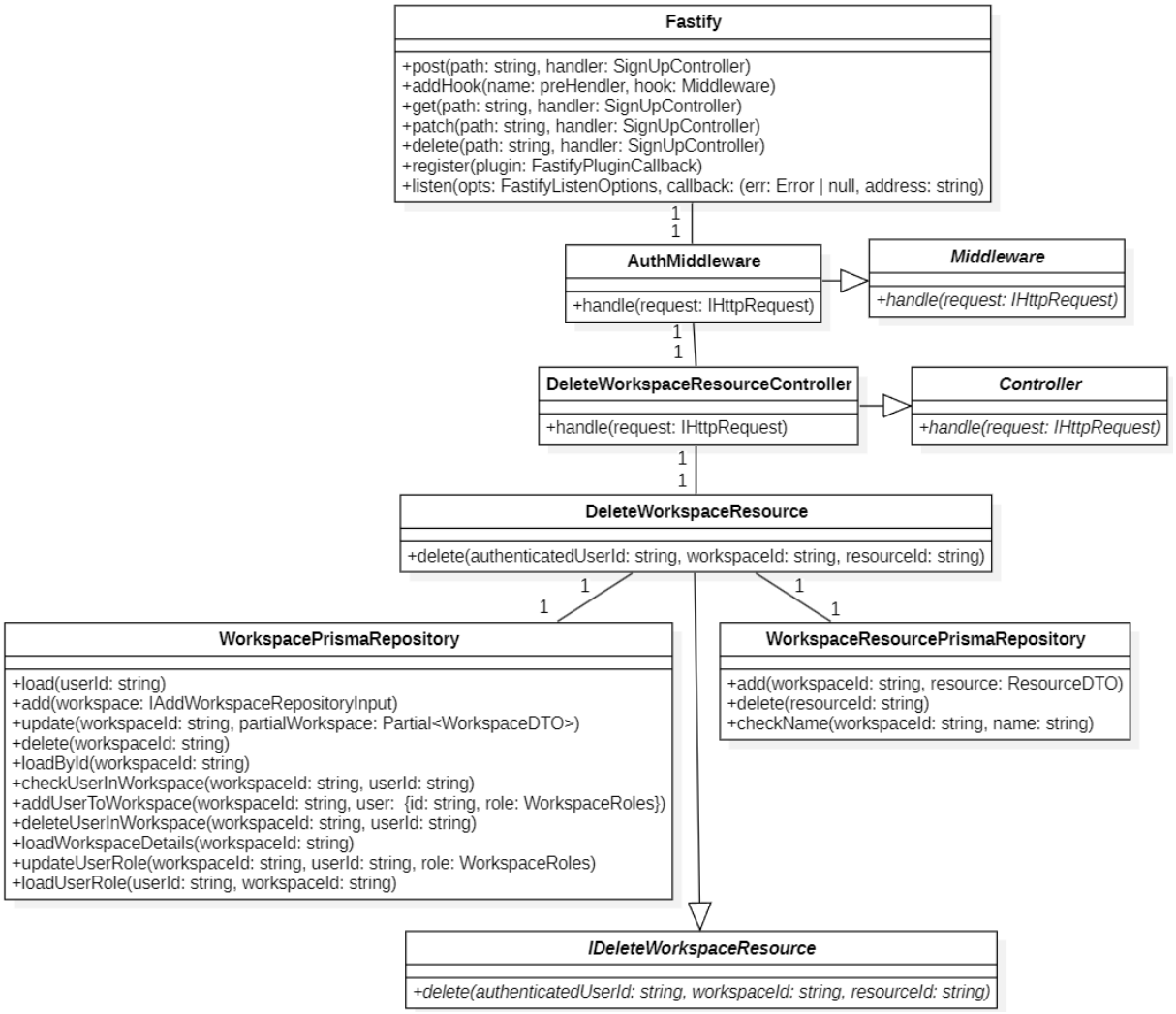


Fonte: Elaborado pelos autores (2025).

A rota para lidar com a obtenção de reservas (Figura 27) é composta por cinco classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54) e *BookingPrismaRepository* (Quadro 84), já foram citadas. As classes que não foram citadas anteriormente são:

- *GetBookingsController:* É o controlador específico para lidar com a obtenção de reservas (Figura 27).
- *LoadBookings*: É a classe responsável por carregar as reservas (Figura 27).
- *ILoadBookings:* É a interface para carregamento de reservas (Figura 27).

Figura 28 - Diagrama de classe do DeleteWorkspaceResource

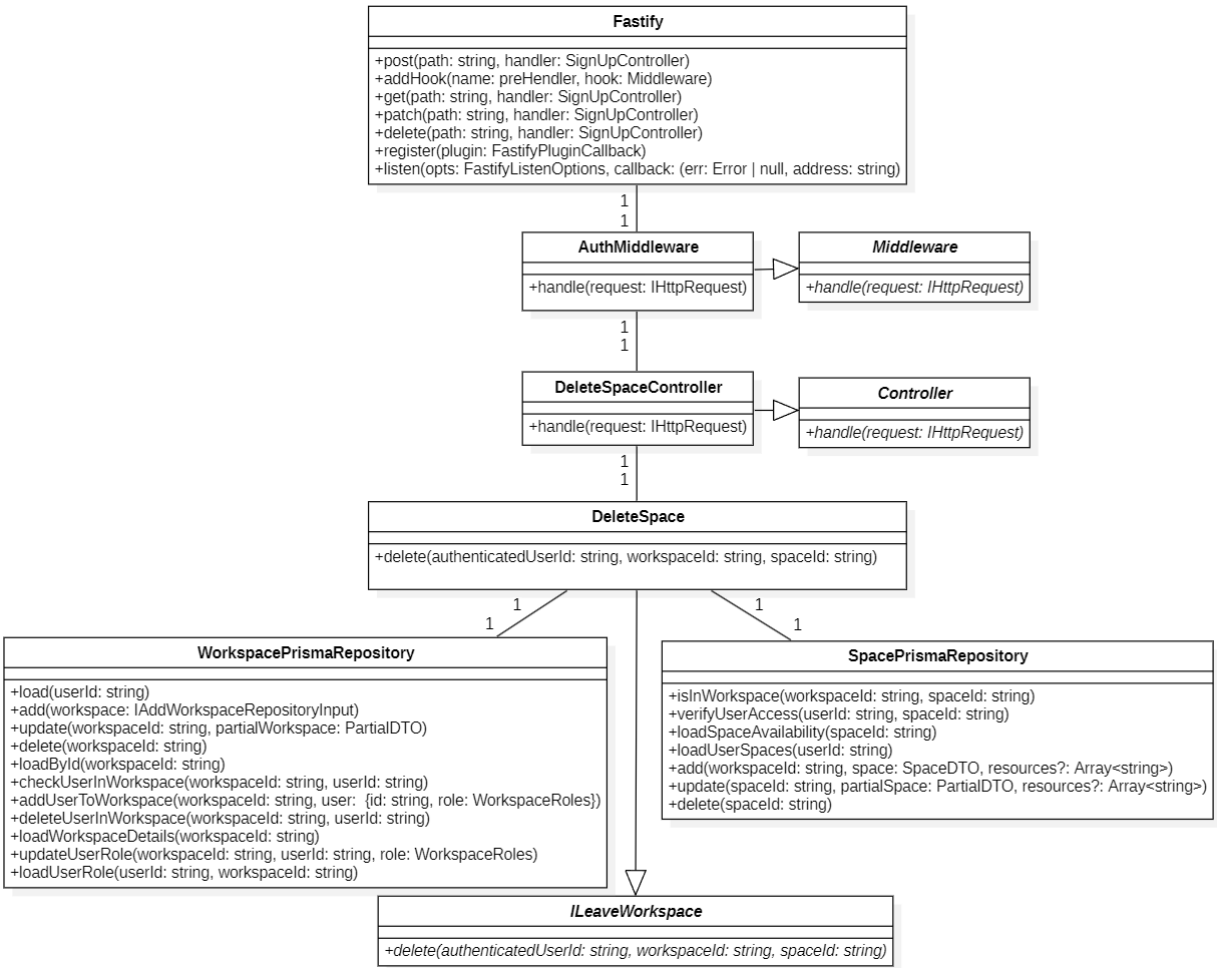


Fonte: Elaborado pelos autores (2025).

A rota para lidar com a exclusão de recursos do *workspace* (Figura 28) é composta por seis classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), *Middleware* (Quadro 54), *WorkspacePrismaRepository* (Quadro 59) e *WorkspaceResourcePrismaRepository* (Quadro 78), já foram citadas. As classes que não foram citadas anteriormente e fazem parte são:

- *DeleteWorkspaceResourceController*: É o controlador específico para lidar com a exclusão de recursos do *workspace* (Figura 28).
- *DeleteWorkspaceResource*: É a classe responsável por deletar recursos do *workspace* (Figura 28).
- *IDeleteWorkspaceResource*: É a interface para a exclusão de recursos de *workspace* (Figura 28).

Figura 29 - Diagrama de classe do DeleteSpace



Fonte: Elaborado pelos autores (2025).

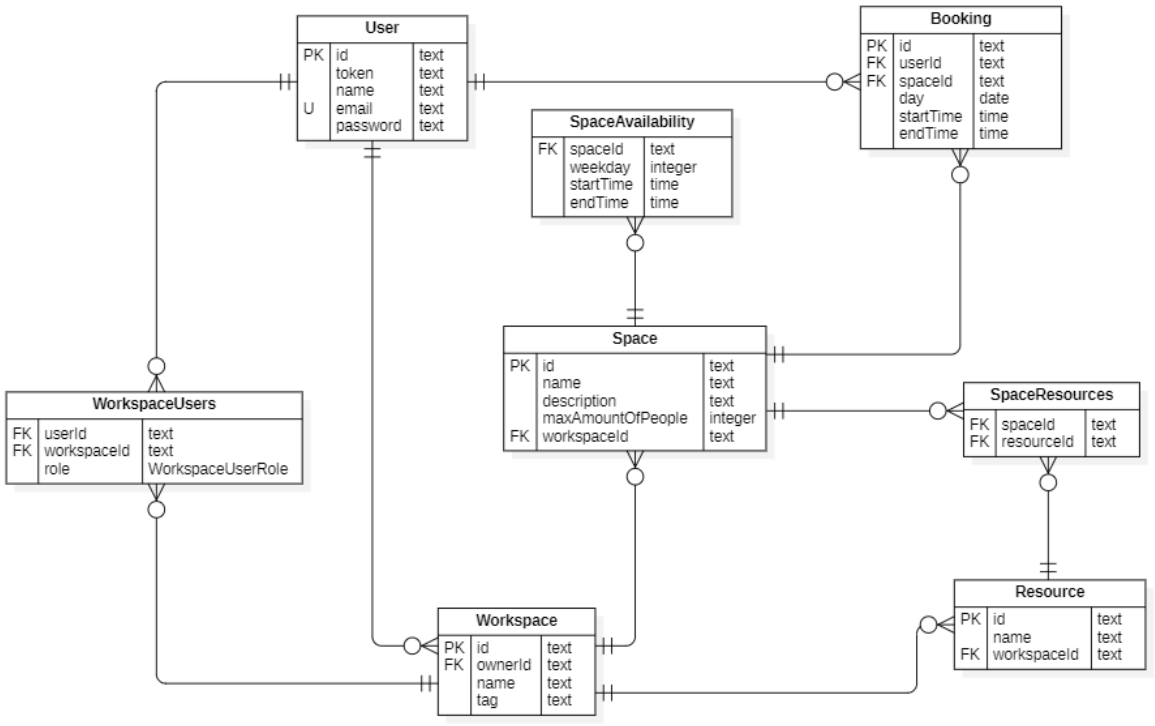
A rota para lidar com a exclusão de recursos de espaços (Figura 29) é composta por seis classes, sendo que, *Fastify* (Quadro 41), *Controller* (Quadro 43), *AuthMiddleware* (Quadro 53), Middleware (Quadro 54), *WorkspacePrismaRepository* (Quadro 59) e *SpacePrismaRepository* (Quadro 81), já foram citadas. As classes que não foram citadas anteriormente são:

- *DeleteSpaceController*: É uma classe responsável por manipular requisições HTTP relacionadas à exclusão de espaços (Figura 29).
- *DeleteSpace*: É a classe responsável porque contém a lógica principal para excluir um espaço dentro de um *workspace* (Figura 29).
- *IDeleteSpace:* É a interface que define o contrato para a funcionalidade de exclusão de espaço (Figura 29).

3.5 Diagrama de Entidade e Relacionamento

Diagrama Entidade Relacionamento é um modelo diagramático que descreve o modelo de dados de um sistema com alto nível de abstração. Ele é a principal representação do Modelo de Entidades e Relacionamentos. Sua maior aplicação é visualizar o relacionamento entre tabelas de um banco de dados, no qual as relações são construídas através da associação de um ou mais atributos destas tabelas (Sommerville, 2019). Figura 30 apresenta o DER do sistema proposto.

Figura 30 - Diagrama de entidade relacionamento.



Fonte: Elaborado pelos autores (2025).

3.6 Dicionário de Dados

O Dicionário de Dados consiste numa lista organizada de todos os elementos de dados que são pertinentes ao sistema. As tabelas devem conter os seguintes campos: Entidade, Atributo, Classe, Domínio, Tamanho e Descrição. O Quadro 20 apresenta o DD da entidade *User*.

Quadro 20 – Dicionário de Dados da entidade User.

Entidade: User				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
id	Determinante	Texto	36	Identificador do usuário
token	Simples	Texto	170	Token de autenticação do usuário
name	Simples	Texto	70	Nome do usuário
email	Simples	Texto	90	Email do usuário
password	Simples	Texto	60	Senha do usuário criptografada

Fonte: Elaborado pelos autores (2025).

O Quadro 21 apresenta o DD da entidade *Workspace*.

Quadro 21 – Dicionário de Dados da entidade Workspace.

Entidade: Workspace				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
id	Determinante	Texto	36	Identificador da *workspace*
ownerId	Determinante	Texto	36	Identificador do dono da *workspace*
name	Simples	Texto	50	Nome da *workspace*
tag	Simples	Texto	5	Marcação e filtro

Fonte: Elaborado pelos autores (2025).

O Quadro 22 apresenta o DD da entidade *WorkspaceUsers*.

Quadro 22 – Dicionário de Dados da entidade WorkspaceUsers.

Entidade: WorkspaceUsers				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
userId	Determinante	Texto	36	Identificador dos usuários
workspaceId	Determinante	Texto	36	Identificador da *workspace*
role	Simples	Enumerado	4	Cargo do usuário na *workspace*

Fonte: Elaborado pelos autores (2025).

O Quadro 23 apresenta o DD da entidade *Space*.

Quadro 23 – Dicionário de Dados da entidade Space.

Entidade: Space				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
id	Determinante	Texto	36	Identificador do espaço
name	Simples	Texto	50	Nome do espaço
description	Simples	Texto	255	Descrição do espaço
maxAmountOfPeople	Simples	Numérico	2	Quantidade máxima de pessoas
workspaceId	Determinante	Texto	36	Identificador da *workspace*

Fonte: Elaborado pelos autores (2025).

O Quadro 24 apresenta o DD da entidade *SpaceAvailability*.

Quadro 24 – Dicionário de Dados da entidade SpaceAvailability.

Entidade: SpaceAvailability				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
spaceId	Determinante	Texto	36	Identificador do espaço
weekday	Simples	Numérico	2	Dia da semana (0 – domingo, 1 – segunda...)
startTime	Simples	Hora	8	Horário de início
endTime	Simples	Hora	8	Horário de término

Fonte: Elaborado pelos autores (2025).

O Quadro 25 apresenta o DD da entidade *SpaceResources*.

Quadro 25 – Dicionário de Dados da entidade SpaceResources.

Entidade: SpaceResources				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
spaceId	Determinante	Texto	36	Identificador dos espaços
resourceId	Determinante	Texto	36	Identificador dos recursos

Fonte: Elaborado pelos autores (2025).

O Quadro 26 apresenta o DD da entidade *Resource*.

Quadro 26 – Dicionário de Dados da entidade Resource.

Entidade: Resource				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
id	Determinante	Texto	36	Identificador do recurso
name	Simples	Texto	20	Nome do recurso
workspaceId	Determinante	Texto	36	Identificador da *workspace*

Fonte: Elaborado pelos autores (2025).

O Quadro 27 apresenta o DD da entidade *Booking*.

Quadro 27 – Dicionário de Dados da entidade Booking.

Entidade: Booking				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
id	Determinante	Texto	36	Identificador da reserva
userId	Determinante	Texto	36	Identificador do usuário
spaceId	Determinante	Texto	36	Identificador do espaço
day	Simples	Data	4	Dia da reserva
startTime	Simples	Hora	8	Horário de início
endTime	Simples	Hora	8	Horário de término

Fonte: Elaborado pelos autores (2025).

4 DESENVOLVIMENTO

Para o desenvolvimento utilizamos o método Scrum que é um *framework* ágil de gestão de projetos que organiza o trabalho em ciclos chamadas de *sprints*, cada uma com duração de duas a quatro semanas, onde as tarefas são selecionadas do *backlog* do produto. As equipes se reúnem diariamente em reuniões breves chamadas *daily scrum* para compartilhar o progresso e identificar obstáculos. No final de cada *sprint*, ocorre a *Sprint Review*, uma revisão para demonstrar o trabalho realizado e obter *feedback*, seguida da *Sprint Retrospective*, uma retrospectiva para refletir sobre o processo. Com papéis definidos, incluindo *Product Owner*, Scrum Master e equipe de desenvolvimento, o Scrum enfatiza a colaboração, adaptação contínua e entrega de valor ao cliente, tornando-o uma abordagem eficaz para o desenvolvimento de produtos complexos em ambientes dinâmicos (FIA, 2025).

Durante o desenvolvimento do projeto, a equipe adaptou o método Scrum para atender às especificidades do ambiente acadêmico. O *Product Owner* foi substituído por um esforço coletivo na elaboração do *backlog*. Em vez de um único responsável pela priorização e gestão do *backlog*, todo o grupo participou dessa tarefa, enquanto o professor monitorava o progresso em cada *sprint*, fornecendo *feedbacks* de melhoria, mas sem se envolver no desenvolvimento ou na gestão do *backlog*. As *sprints* tiveram uma duração de duas semanas, tendo reuniões apenas nas segundas, sábados e ocasionalmente em outros dias. Quanto a função de cada membro:

- Julio Cesar desempenhou o papel de Scrum Master, acumulando também a função de desenvolvedor;
- Rogério Henrique atuou como testador, garantindo a qualidade do aplicativo e colaborou com a documentação do projeto;
- Rafael Cantovitz foi responsável pelo design e documentação do projeto.

Essas adaptações permitiram uma abordagem colaborativa e eficaz, adequada ao contexto educacional e ao objetivo de aprendizagem dos participantes.

4.1 Etapas de Desenvolvimento

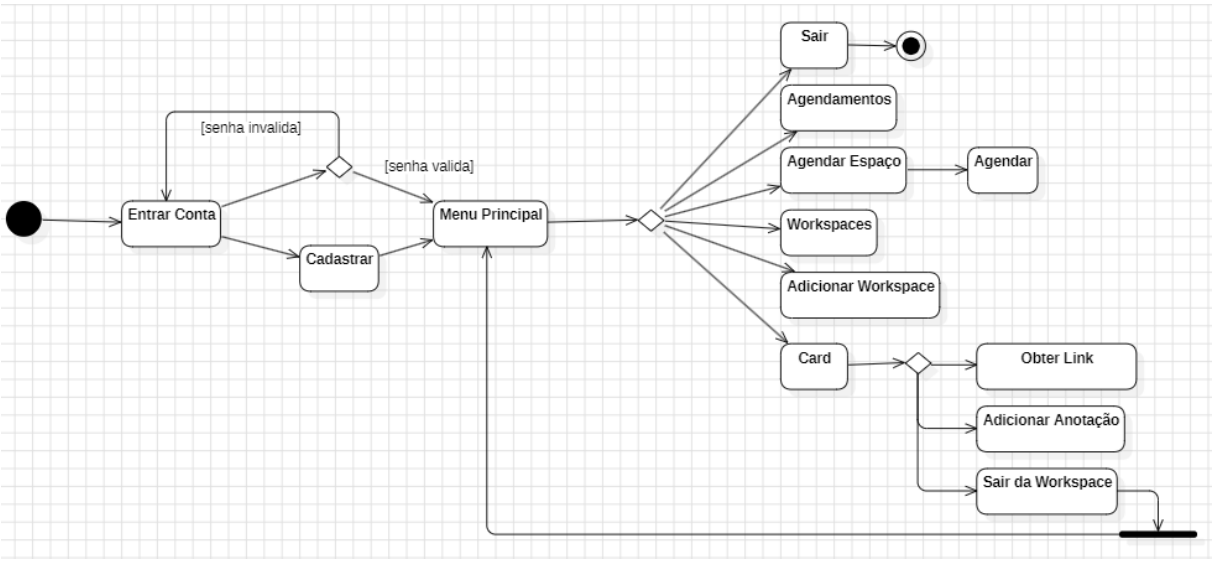
Durante o semestre, o desenvolvimento do projeto foi organizado em cinco sprints quinzenais, seguindo os princípios da metodologia ágil Scrum. A seguir, detalham-se as atividades realizadas em cada ciclo de desenvolvimento:

- *Sprint 1*: teve como objetivo iniciar o projeto, com a definição das telas iniciais e das funcionalidades essenciais. Foram especificados os requisitos, escolhidas as ferramentas de desenvolvimento e implementadas a tela de login, a tela de cadastro e a tela inicial do sistema. Também foi criada a API de autenticação de usuários, integrando o *front-end* com o *back-end* de forma segura.
- *Sprint 2*: o foco foi expandir as funcionalidades da aplicação, com ênfase no gerenciamento de usuários dentro das *workspaces*. Foram implementadas as operações de adição e remoção de membros, a funcionalidade para que usuários se desvinculem de uma *workspace*, além da separação entre usuários padrão, gestores e proprietários, estruturando o sistema de permissões baseado em RBAC.
- *Sprint 3*: concentrou-se no desenvolvimento das funcionalidades de gerenciamento de espaços. Foram implementadas as ações de adicionar, editar e excluir ambientes, bem como a definição de atributos como capacidade máxima e recursos disponíveis. Essa etapa consolidou a base para o funcionamento do sistema de agendamento.
- *Sprint 4*: envolveu a adição dos horários de disponibilidade aos formulários de cadastro e a criação da tela de agendamento. Usuários passaram a poder selecionar datas e horários com base na disponibilidade registrada, exigindo ajustes nas regras de validação.
- *Sprint 5*: o objetivo foi finalizar as funcionalidades principais e melhorar a experiência do usuário. Foram implementadas a visualização de reservas e melhorias na responsividade da interface. A sprint também incluiu testes internos com usuários fictícios e revisões técnicas conduzidas pela equipe para validação das entregas.

4.2 Interfaces de Usuário

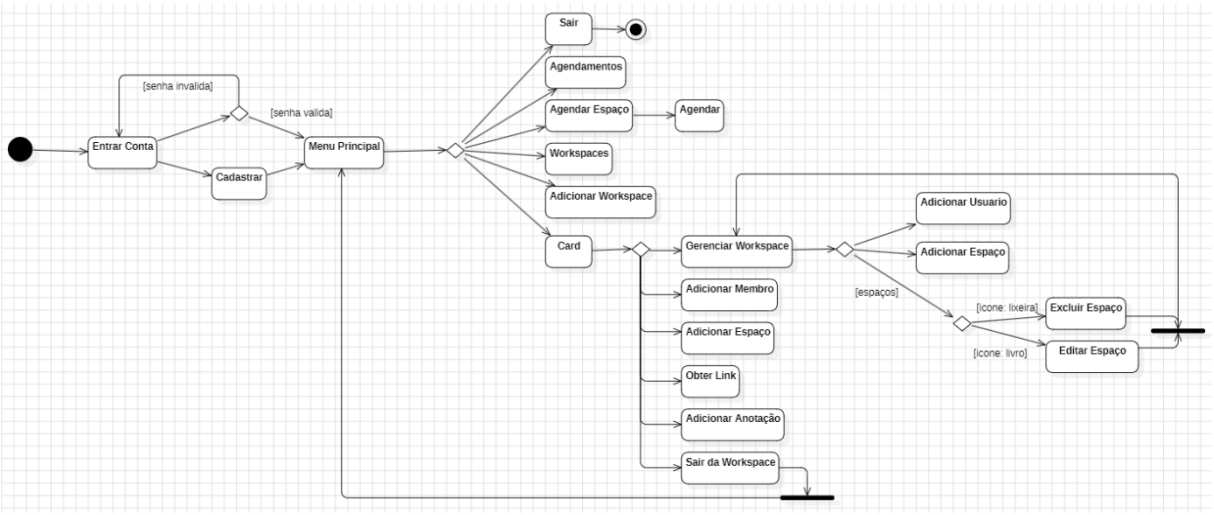
A necessidade da construção de uma interface amigável ao usuário é fundamental em um sistema. A interface faz parte do sistema computacional e determina como as pessoas operam e controlam o sistema. Quando uma interface é bem projetada, ela é compreensível, agradável e controlável. Neste contexto, estes protótipos têm como objetivo apresentar o aplicativo e os recursos da tela (Figura 31, Figura 32 e Figura 33).

Figura 31 – Diagrama de fluxo (mapa das telas do usuário).



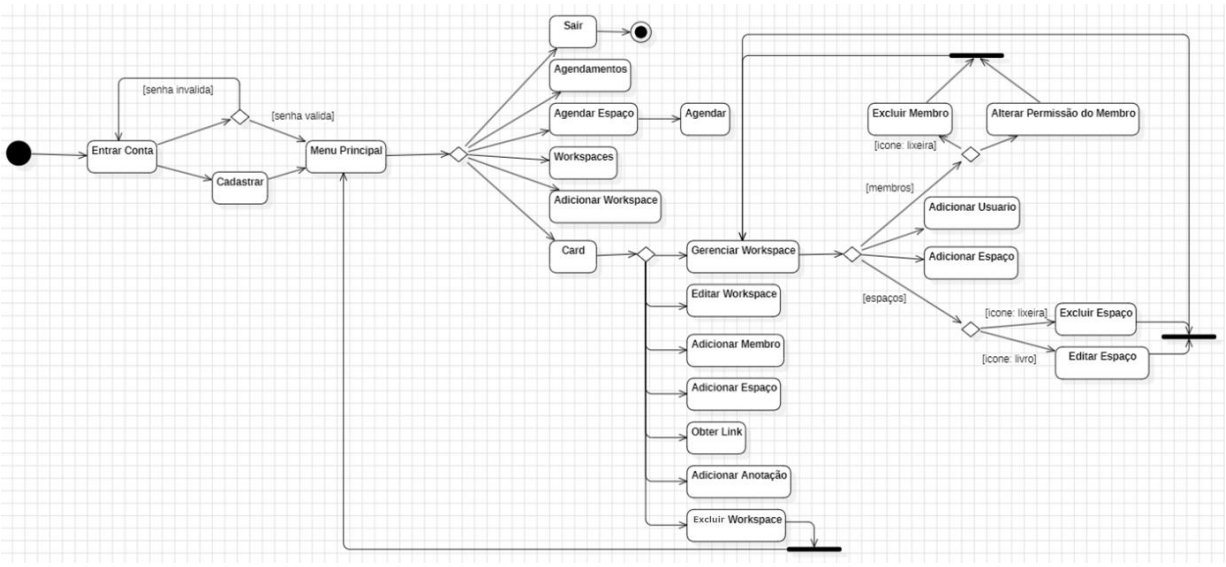
Fonte: Elaborado pelos autores (2025).

Figura 32 – Diagrama de fluxo (mapa das telas do gerente).



Fonte: Elaborado pelos autores (2025).

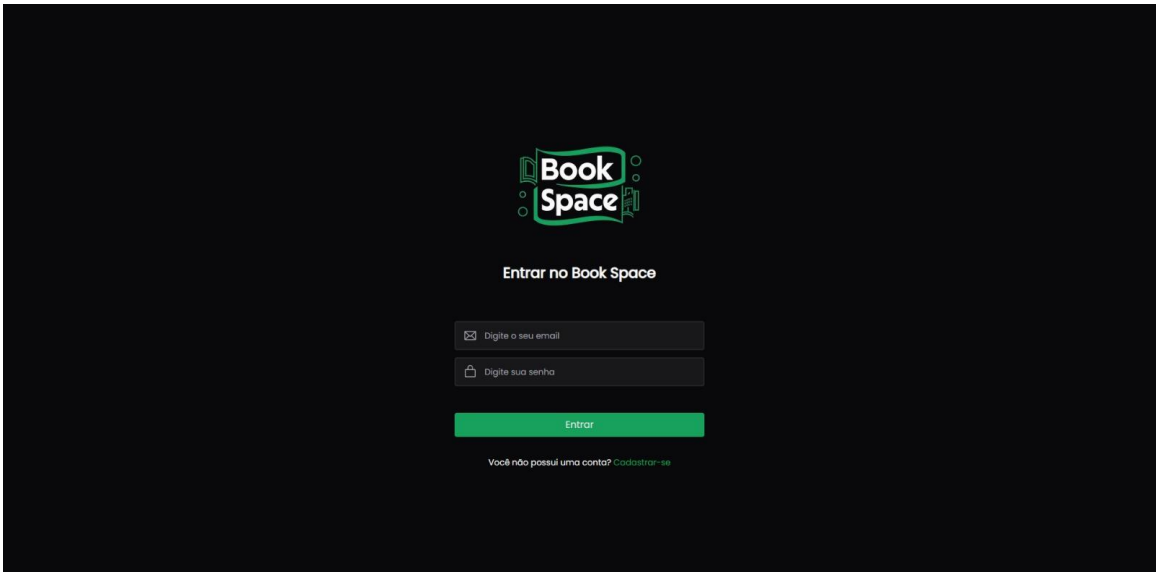
Figura 33 – Diagrama de Fluxo (mapa das telas do dono).



Fonte: Elaborado pelos autores (2025).

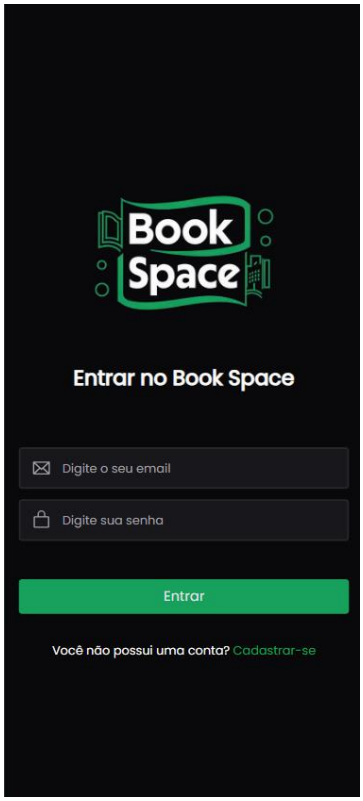
A Figura 34 e Figura 35 apresentam a página de *Login*, onde se consegue entrar no site, e ter privilégios de usuário, colocando o e-mail e senha pré-cadastrados.

Figura 34 - Captura da página de login



Fonte: Elaborado pelos autores (2025).

Figura 35 - Captura da página de login (Mobile)



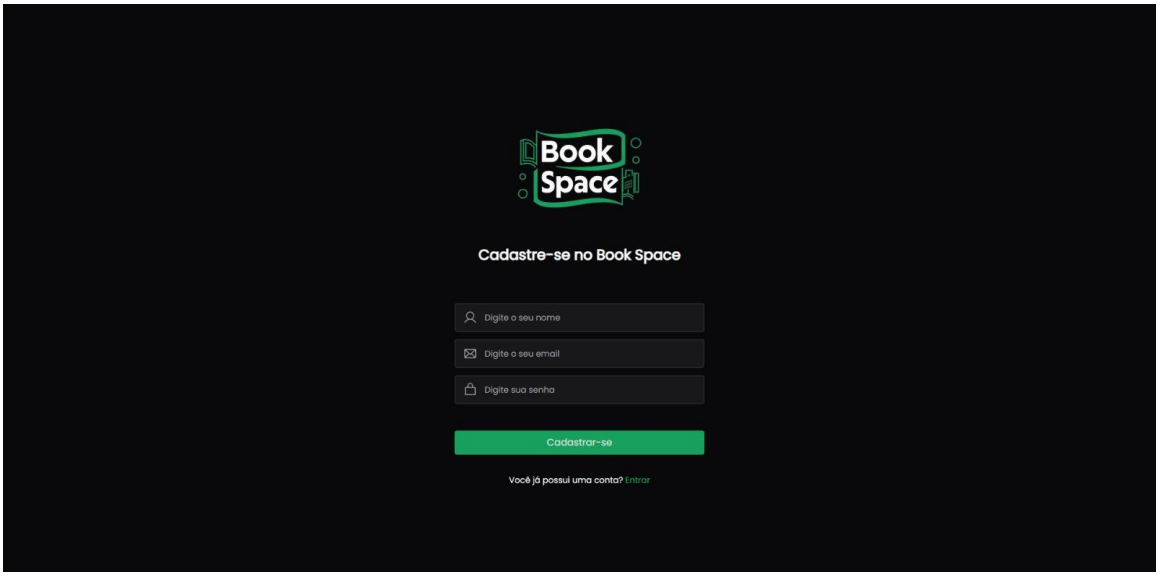
Fonte: Elaborado pelos autores (2025).

A página de login apresentada na Figura 34 e Figura 35 é composta por:

- **Campo E-mail:** Para colocar seu e-mail antes cadastrado.
- **Campo Senha:** Para colocar sua senha antes cadastrado.
- **Botão Entrar:** Para conseguir entrar no sistema.
- **Link Cadastrar-se:** Permite que o usuário possa ir direto para a tela de cadastro para se cadastrar.

A Figura 36 e Figura 37 apresentam a página de cadastro, onde se completa essas informações para ter acesso ao sistema com e-mail e senha, sendo todas informações obrigatórias.

Figura 36 - Captura da página de cadastro.



Fonte: Elaborado pelos autores (2025).

Figura 37 - Captura da página de cadastro (Mobile).

Fonte: Elaborado pelos autores (2025).

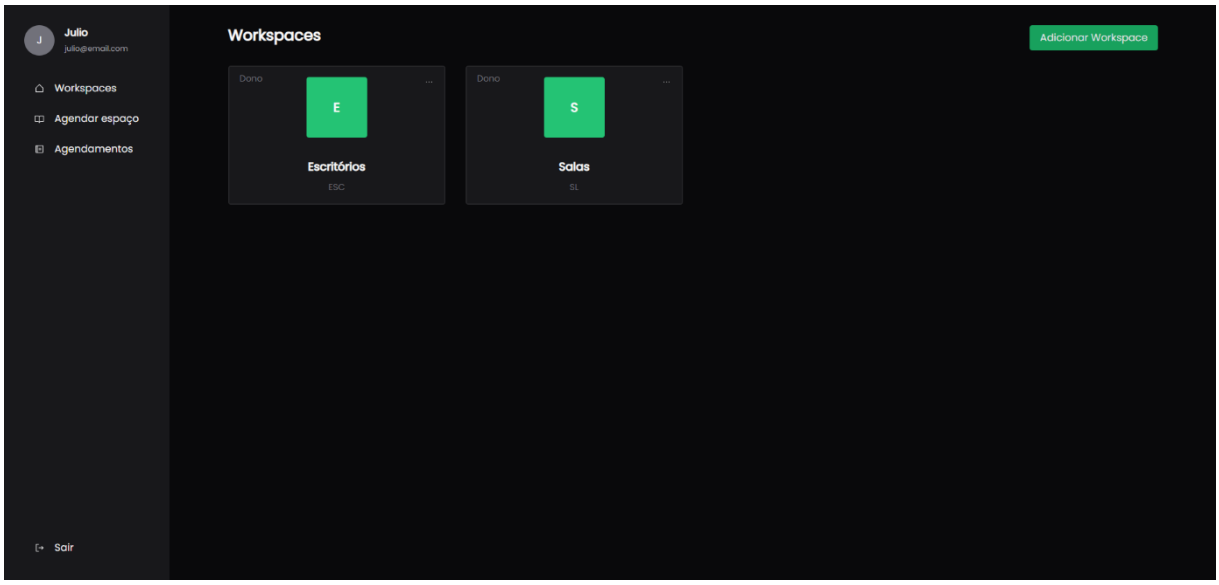
A página de cadastro apresentada na Figura 36 e Figura 37 é composta por:

- **Campo Nome:** Para colocar o nome.

- **Campo E-mail:** Para colocar seu e-mail.
- **Campo senha:** Para colocar uma senha.
- **Botão Cadastrar-se:** Ao clicar nesse botão, salva suas informações declaradas, e podendo ter a aprovação de um futuro login para entrar no sistema.
- **Link “Entrar”:** Permite que o usuário possa ir direto à tela de login se já tiver um cadastro.

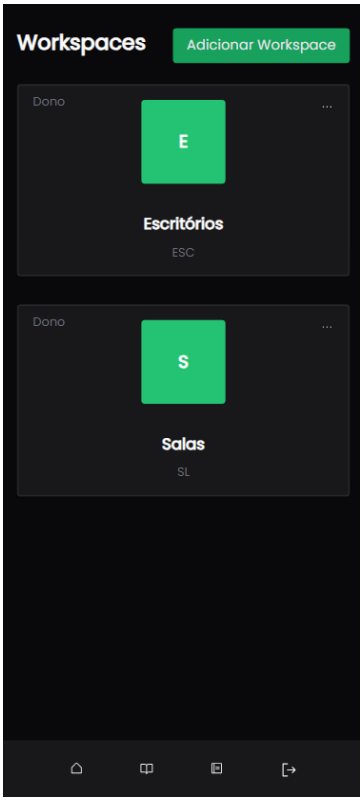
A Figura 38 e Figura 39 apresentam a página principal, onde se encontra as abas para ter acesso as “Workspaces”, para poder agendar um espaço e para listar os agendamentos. Além de conter um botão para criar uma *workspace* e um botão para sair da conta.

Figura 38 - Captura da página principal.



Fonte: Elaborado pelos autores (2025).

Figura 39 - Captura da página principal (Mobile).



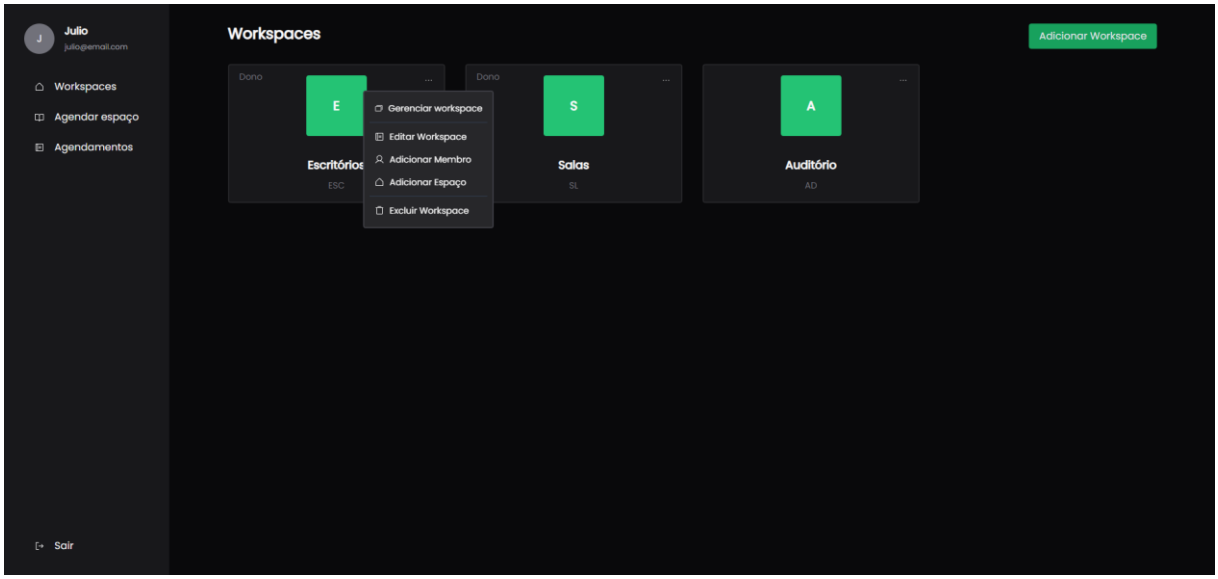
Fonte: Elaborado pelos autores (2025).

A página principal apresentada na Figura 38 e Figura 39 é composta por:

- **Botão Workspaces:** Aba que contém as *workspaces*.
- **Botão Agendar espaço:** Aba para requisitar um agendamento de espaço.
- **Botão Agendamentos:** Aba para listar os agendamentos realizados.
- **Botão Adicionar Workspace:** Ao clicar nesse botão, dá início a criação de uma nova *workspace*.
- **Botão Escritórios (workspace):** Abre a *workspace*.
- **Botão Configurações da workspace:** Abre um menu de configurações da *workspace*.
- **Botão Sair:** Ao clicar nesse botão, retorna para a tela de login.

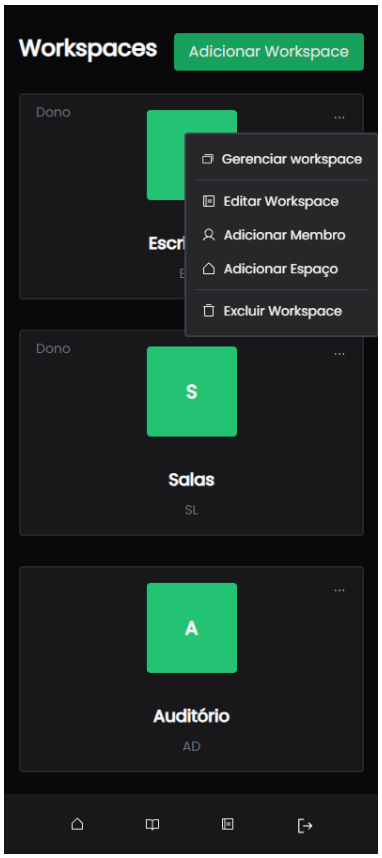
A Figura 40 e a Figura 41 apresentam a tela da página principal, mostrando o menu de configurações da *workspace*, em que se encontra todas as funções que o dono pode exercer na respectiva *workspace*.

Figura 40 - Captura da tela da página principal mostrando as funções do dono.



Fonte: Elaborado pelos autores (2025).

Figura 41 - Captura da tela da página principal mostrando as funções do dono (Mobile).



Fonte: Elaborado pelos autores (2025).

A tela da página principal da *workspace* com as funções do dono, apresentada na Figura 40 e Figura 41, é composta por:

- **Campo Gerenciar Workspace:** Abre uma nova tela para gerenciar aquela *workspace*.
- **Campo Editar Workspace:** Abre uma nova tela para mudar o nome e a *tag* da *workspace*.
- **Campo Adicionar Membro:** Abre uma tela para poder adicionar um novo membro a *workspace*.
- **Campo Adicionar Espaço:** Abre uma nova tela para poder adicionar uma nova sala a *workspace*.
- **Campo Excluir Workspace:** Exclui a *workspace* em questão.

O gerente designando pelo criador da *workspace* tem o mesmo tipo de interface, porém apresenta opções diferentes das do dono da *workspace*. A tela da página principal da *workspace* com as funções do gerente, é composta por:

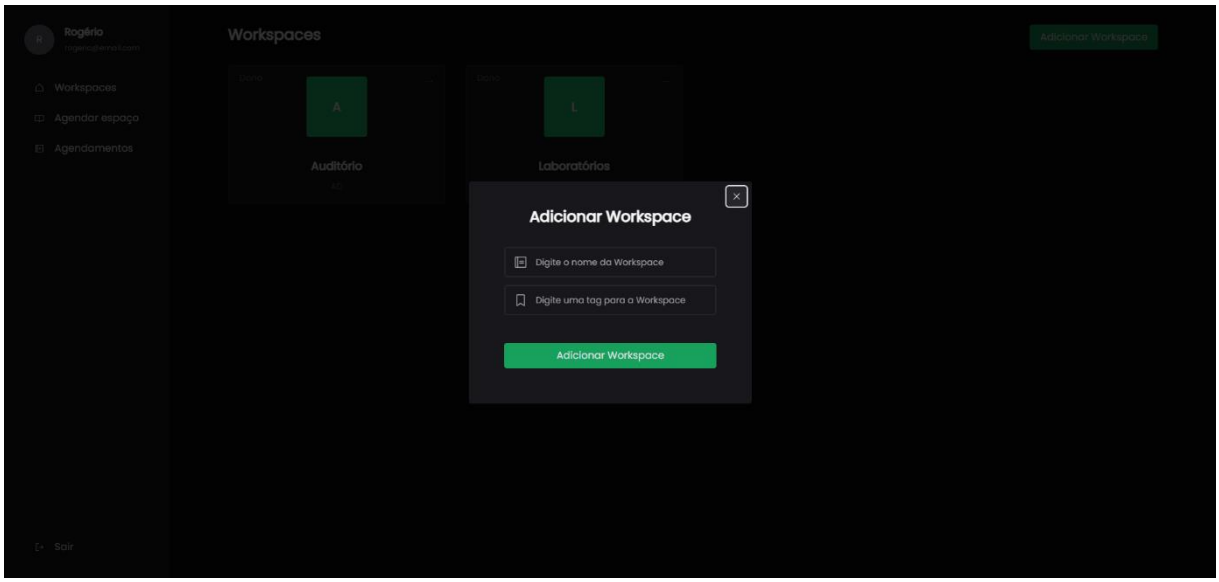
- **Campo Gerenciar Workspace:** Abre uma nova tela para gerenciar aquela *workspace*.
- **Campo Adicionar Membro:** Abre uma tela para poder adicionar um novo membro a *workspace*.
- **Campo Adicionar Espaço:** Abre uma nova tela para poder adicionar uma nova sala a *workspace*.
- **Campo Sair da Workspace:** O usuário gerente pode sair da *workspace* que está participando.

O usuário padrão, quando é atribuído a uma *workspace* tem o mesmo tipo de interface, porém apresenta opções diferentes das do dono e do gerente da *workspace*. O menu de configurações da *workspace*, é composta por:

- **Botão Sair da Workspace:** O usuário sai da *workspace*.

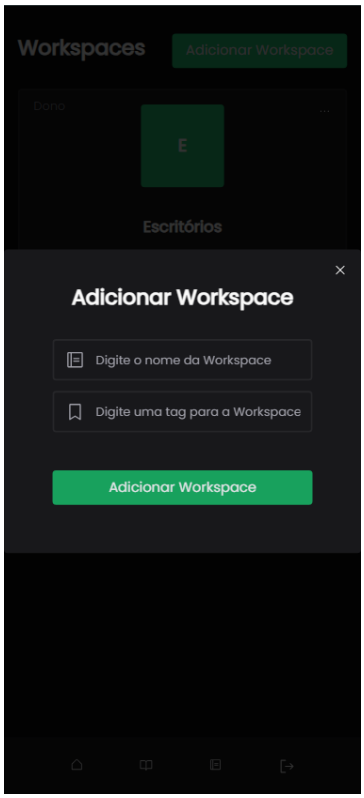
A Figura 42 e Figura 43 apresentam a página de criação de uma *workspace*, onde se deve completar as informações de nome e *tag* para poder criar a *workspace*.

Figura 42 - Captura da página de criação da workspace.



Fonte: Elaborado pelos autores (2025).

Figura 43 - Captura da página de criação da workspace (Mobile).



Fonte: Elaborado pelos autores (2025).

A página de criação da *workspace* apresentada na Figura 42 e Figura 43 é composta por:

- **Campo Nome da Workspace:** Para colocar o nome.
- **Campo tag da Workspace:** Para colocar a *tag*.

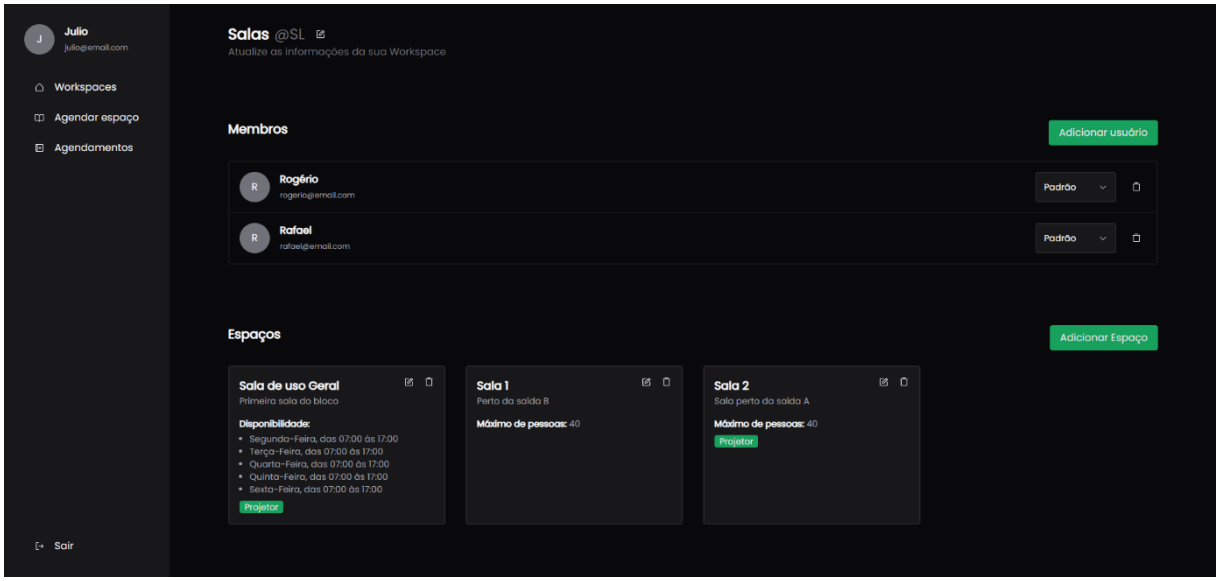
A página focada na edição apresenta aspectos parecidos com a tela de criação da *workspace*, ela permite mudar coisas básicas.

A tela de edição da *workspace*, é composta por:

- **Campo Nome:** Para mudar o nome.
- **Campo Tag:** Para mudar a *tag*.
- **Botão Editar Workspace:** Salva as mudanças feitas nos outros campos.

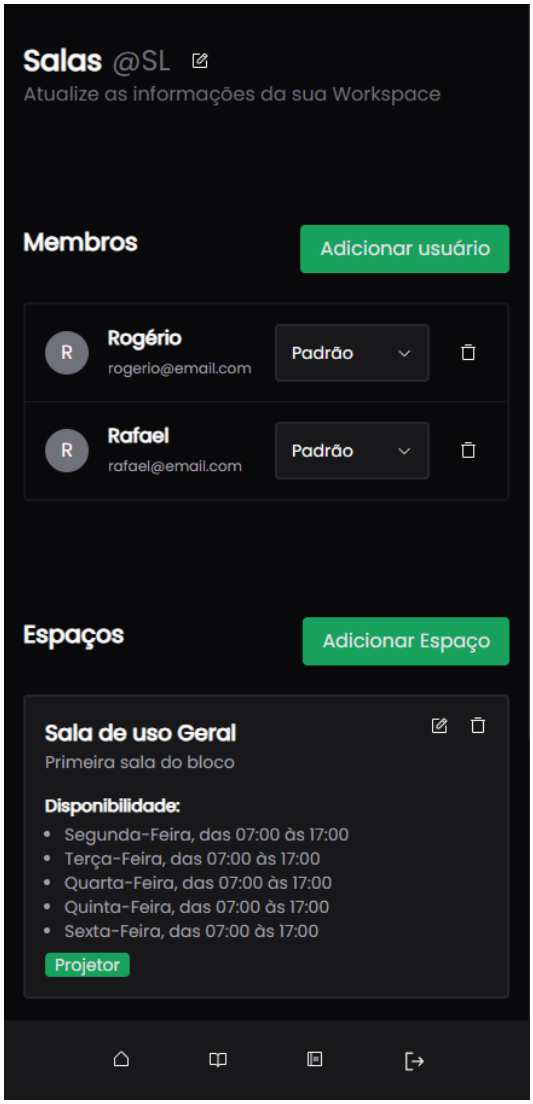
A Figura 44 e a Figura 45 apresenta a tela de gerenciamento da *workspace*, mostrando as funções que o dono pode exercer nas suas *workspaces*.

Figura 44 - Captura da tela de gerenciamento da workspace como dono.



Fonte: Elaborado pelos autores (2025).

Figura 45 - Captura da tela de gerenciamento da workspace como dono (Mobile).



Fonte: Elaborado pelos autores (2025).

A tela de gerenciamento da *workspace* com as funções do dono, apresentada na Figura 44 e Figura 45, é composta por:

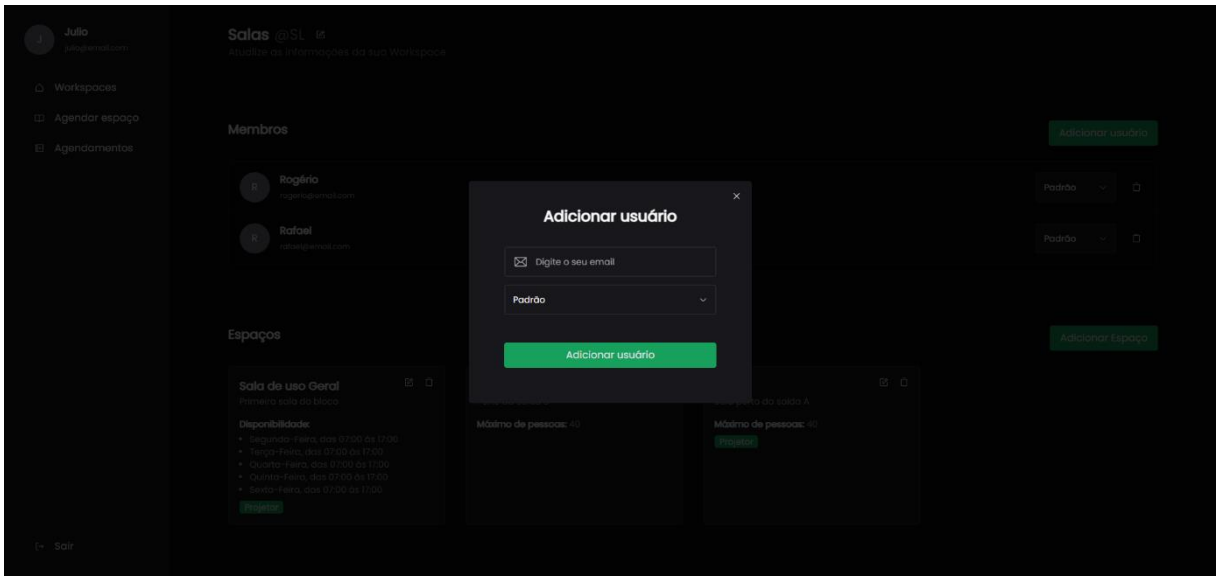
- **Botão Adicionar usuário:** Abre uma tela para poder adicionar um novo usuário a *workspace*.
- **Campo de modificação de cargo:** Permite escolher a função de um usuário específico, assim como excluir um usuário.
- **Botão Adicionar Espaço:** Abre uma nova tela para poder adicionar uma nova sala a *workspace*.
- **Membros**: Mostra quem são os membros.
- **Espaços**: Mostra os espaços dentro da *workspace*.

A tela de gerenciamento da *workspace*, mostrando as funções que o gerente pode exercer nas suas *workspaces*, segue o mesmo padrão da tela do dono, tendo apenas uma mudança. A tela de gerenciamento da *workspace* com as funções do gerente, é composta por:

- **Botão Adicionar usuário:** Abre uma tela para poder adicionar um novo usuário a *workspace*.
- **Campo de modificação de cargo:** Não permite escolher a função de um usuário específico, mas pode excluir um usuário desde que não seja outro gerente ou dono.
- **Botão Adicionar Espaço:** Abre uma nova tela para poder adicionar uma nova sala a *workspace*.
- **Membros**: Mostra quem são os membros.
- **Espaços**: Mostra os espaços dentro da *workspace*.

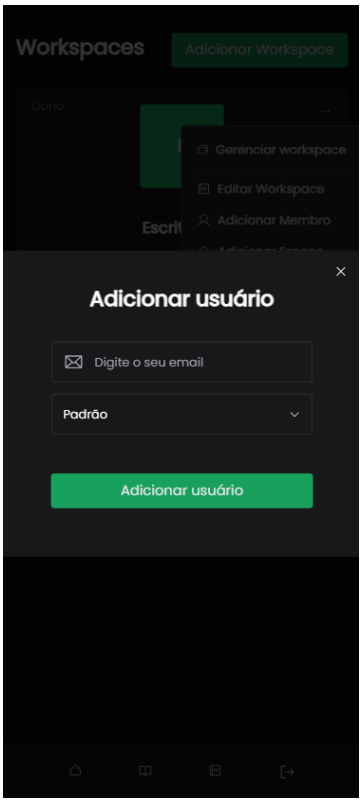
A Figura 46 e a Figura 47 apresentam a tela de adicionar usuário na *workspace*.

Figura 46 - Captura da tela de adicionar usuário na workspace.



Fonte: Elaborado pelos autores (2025).

Figura 47 - Captura da tela de adicionar usuário na workspace (Mobile).



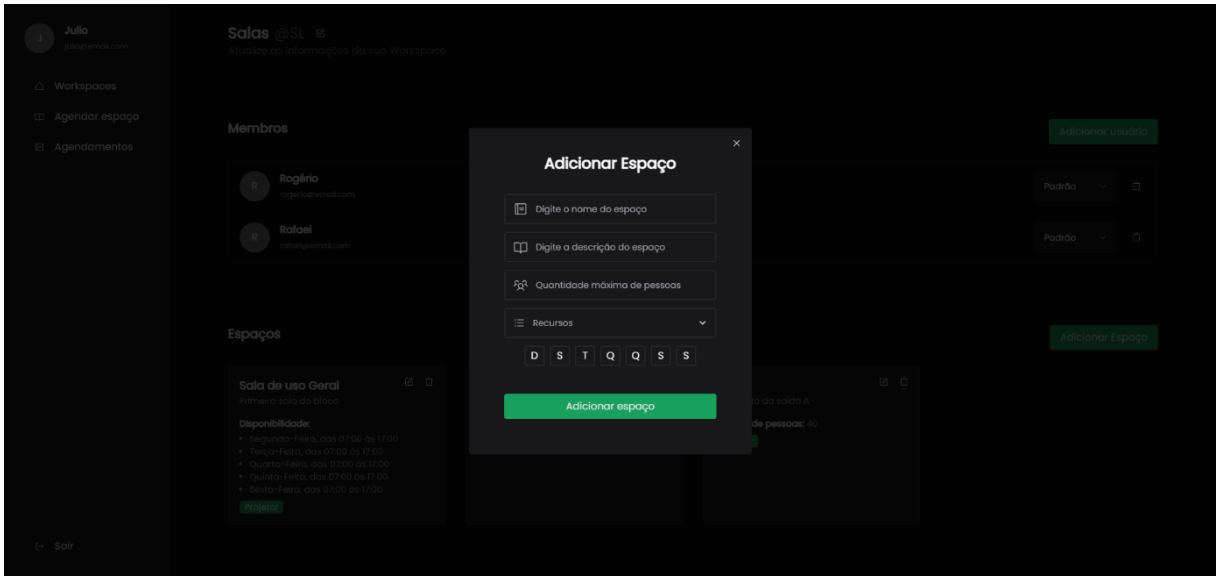
Fonte: Elaborado pelos autores (2025).

A tela de adicionar usuário na *workspace*, apresentada na Figura 46 e Figura 47, é composta por:

- **Campo Digitar o e-mail:** Campo para preencher com o e-mail do membro a adicionar.
- **Campo Selecionar o cargo**: Seleciona o cargo de um usuário com as opções de gerente ou usuário padrão.
- **Botão Adicionar Usuário:** Aplica o envio do e-mail para o usuário a ser adicionado.

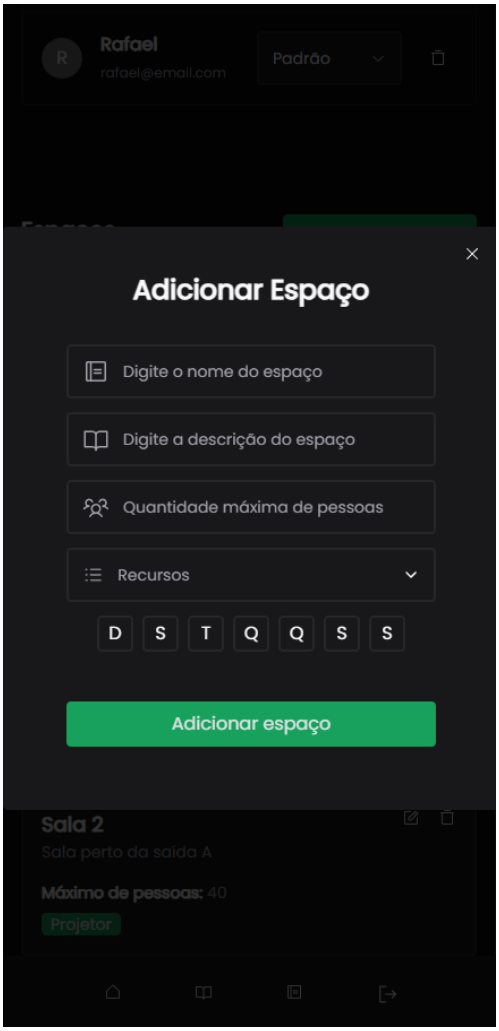
A Figura 48 e a Figura 49 apresentam a tela de um gerente ou administrador fazendo a adição de um espaço ou de uma sala a uma *workspace*.

Figura 48 - Captura da tela de adição de um espaço a uma workspace.



Fonte: Elaborado pelos autores (2025).

Figura 49 - Captura da tela de adição de um espaço a uma workspace (Mobile).



Fonte: Elaborado pelos autores (2025).

A tela do gerente ou administrador na criação de uma sala na *workspace*, apresentada na Figura 48 e Figura 49, é composta por:

- **Campo Nome do Espaços:** Campo para preencher com o nome do espaço a adicionar.
- **Campo Descrição do Espaços:** Campo para preencher com a descrição do espaço a adicionar.
- **Campo Máximo de pessoas do Espaços:** Campo para preencher com máximo de pessoas do espaço a adicionar.
- **Campo Recursos do Espaços:** Campo para preencher com os recursos presentes no espaço a adicionar.

- **Campo Dias:** Seleciona os dias da semana que a sala poderá ser agendada.
- **Botão Adicionar Espaço:** Aplica a criação de um novo espaço dentro da respectiva *workspace*.

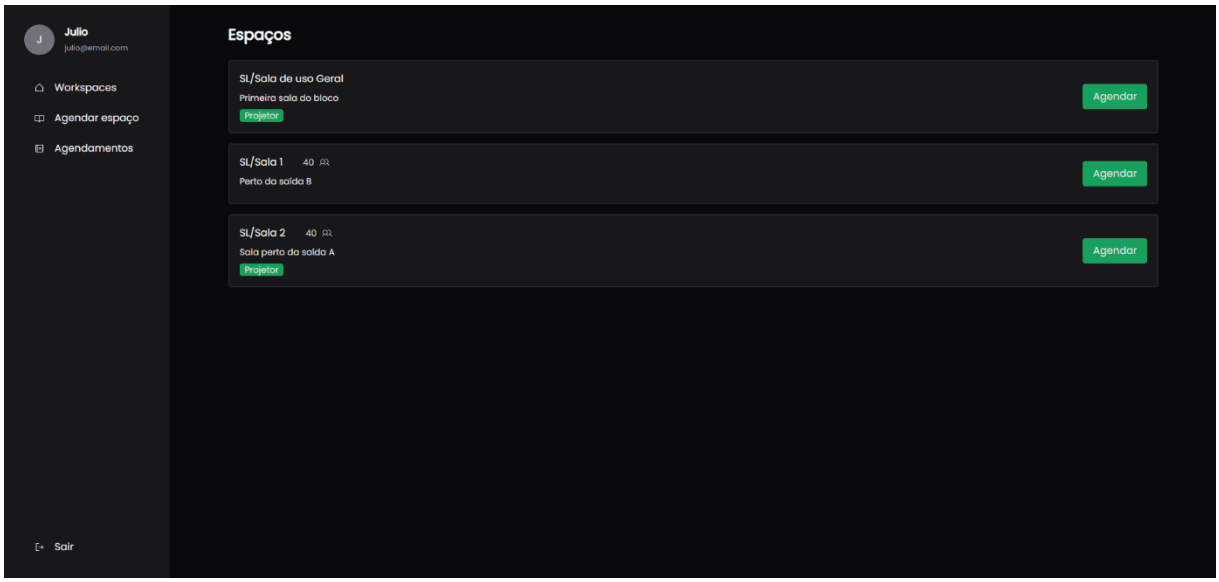
A tela de um gerente ou administrador quando precisa fazer uma edição de um espaço ou de uma sala de uma *workspace* é parecida com a tela de criação.

A tela do gerente ou administrador na edição de uma sala na *workspace*, é composta por:

- **Campo Nome do Espaços:** Campo para alterar o nome de um espaço já adicionado.
- **Campo Descrição do Espaços:** Campo para alterar a descrição de um espaço já adicionado.
- **Campo Máximo de pessoas do Espaços:** Campo para mudar o máximo de pessoas de um espaço já criado.
- **Campo Recursos do Espaços:** Campo para preencher ou remover os recursos que estão presentes no espaço já adicionado.
- **Campo Dias:** Seleciona os dias da semana que a sala poderá ser agendada.
- **Botão Adicionar Espaço:** Aplica a alteração em um espaço dentro da respectiva *workspace*.

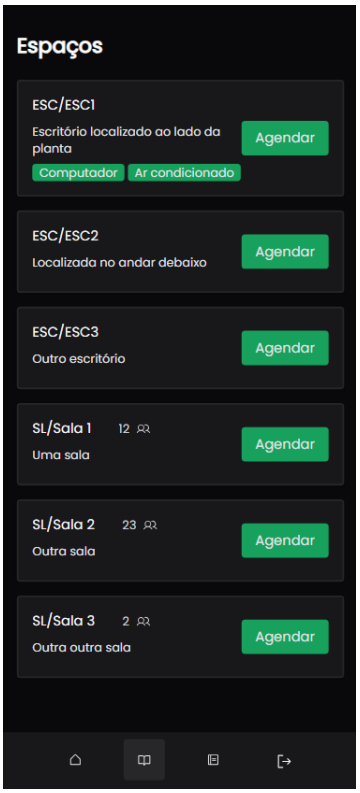
A Figura 50 e a Figura 51 apresentam a tela de espaços, que um usuário comum pode visualizar e agendar.

Figura 50 - Captura da tela espaços.



Fonte: Elaborado pelos autores (2025).

Figura 51 - Captura da tela espaços (Mobile).



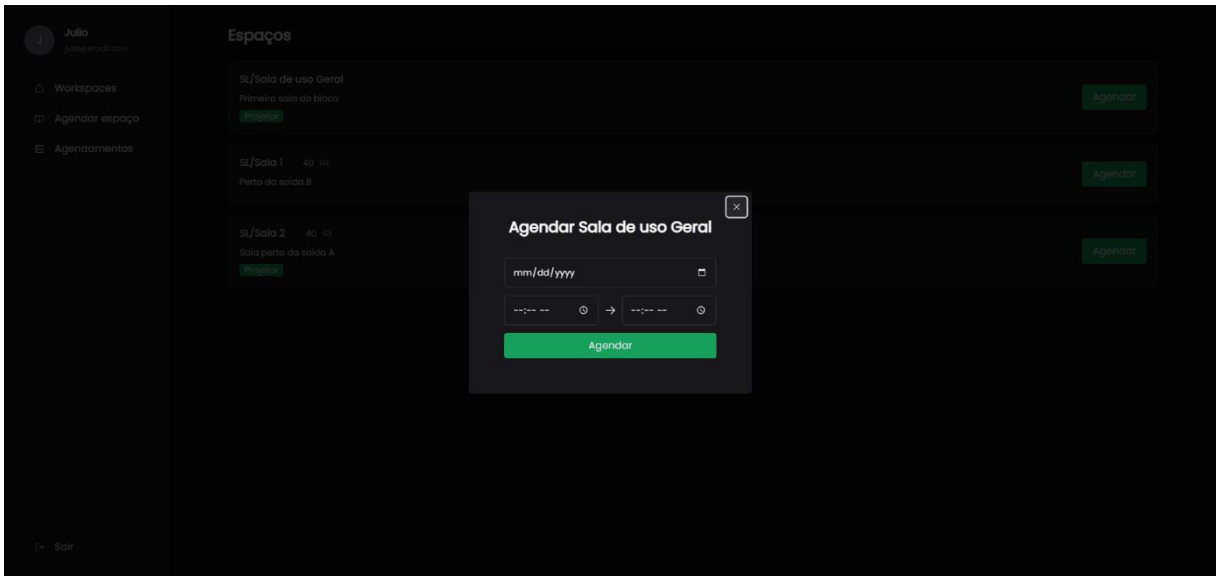
Fonte: Elaborado pelos autores (2025).

A tela espaços, apresentada na Figura 50 e Figura 51, é composta por:

- **Espaços:** Mostra os espaços dentro da *workspace*, podendo já ser agendadas.
- **Botão Agendar:** Abre a tela de agendamento de espaço.

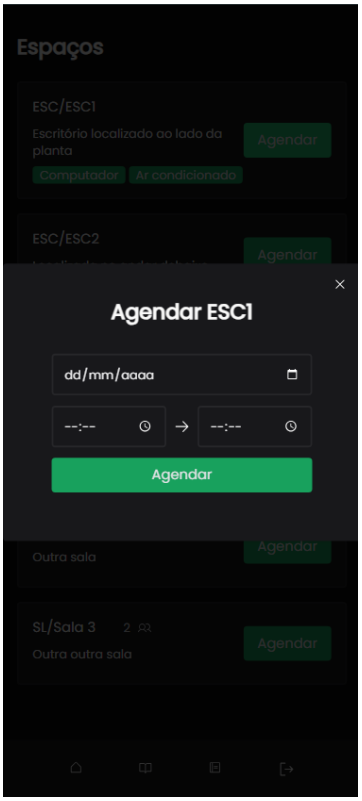
A Figura 52 e a Figura 53 apresentam a tela um usuário padrão selecionando o dia e a hora que a sala ficará agendada, esta tela faz parte do processo de agendamento.

Figura 52 - Captura da tela de agendamento de um espaço.



Fonte: Elaborado pelos autores (2025).

Figura 53 - Captura da tela de agendamento de um espaço (Mobile).



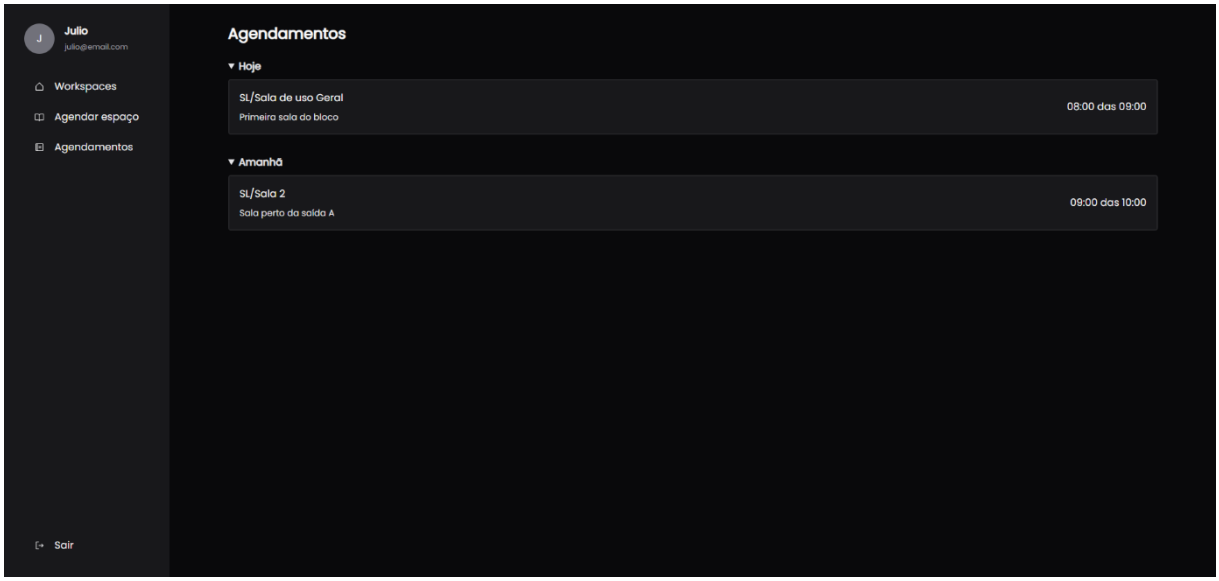
Fonte: Elaborado pelos autores (2025).

A tela de um usuário padrão fazendo o agendamento de uma sala na *workspace*, apresentada na Figura 52 e Figura 53, é composta por:

- **Campo Data:** Seleciona uma data que a sala será agendada
- **Campo Hora Entrada:** Seleciona o horário que a sala estará ocupada.
- **Campo Hora Saida:** Seleciona o horário que a sala está livre para ser agendada novamente.

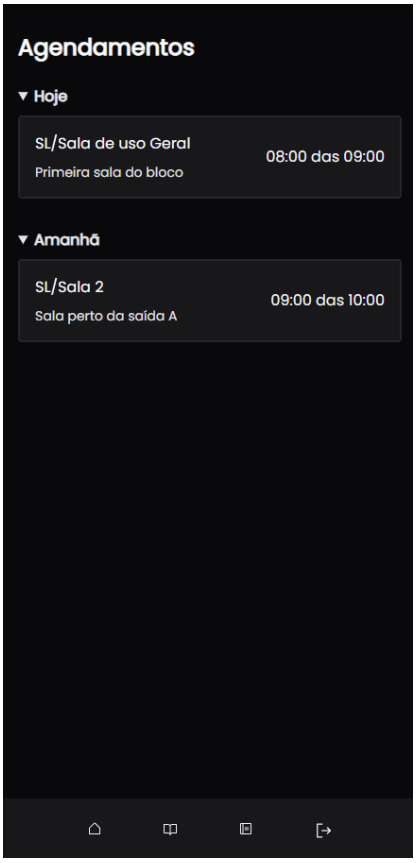
A Figura 54 e a Figura 55 apresentam a tela de agendamentos, onde o usuário pode visualizar os agendamentos que ele realizou.

Figura 54 – Captura da tela de agendamentos.



Fonte: Elaborado pelos autores (2025).

Figura 55 - Captura da tela de agendamentos (Mobile).



Fonte: Elaborado pelos autores (2025).

A tela de agendamentos, apresentada na Figura 54 e Figura 55, é composta por:

- **Datas**: A data em que o espaço foi reservado.
- **Agendamento**: Elemento que contêm as informações do espaço reservado e informações de agendamento, como hora de entrada e hora de saída.

5 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo auxiliar organizações no gerenciamento de espaços, como salas de reunião em empresas ou salas de aula e laboratórios em uma instituição de ensino, através de uma aplicação que simplifique o processo de reserva. A proposta baseou-se na observação prática de problemas frequentes na gestão de ambientes organizacionais, com o intuito de otimizar o gerenciamento desses locais. Para isso, realizou-se uma análise comparativa com outros aplicativos, identificando seus pontos positivos e negativos, visando realizar uma proposta para o problema em questão.

Reconhecendo os desafios naturais da gestão manual de espaços, como a dificuldade em manter a organização de horários e a limitação de acordos informais que podem gerar desencontros, o grupo optou por desenvolver um sistema que busque diminuir isso. A proposta visa automatizar o controle de reservas, minimizar falhas operacionais e aumentar a produtividade dentro das corporações. Por fim, foi detalhado as etapas de desenvolvimento com o uso do método Scrum, realizando a divisão do trabalho por meio de cinco *sprints*, contendo o que foi realizado durante a *sprint,* demonstrando como e quando foi realizada.

Embora a aplicação já ofereça funcionalidades que auxiliam no gerenciamento eficiente de espaços, há um grande potencial para futuras melhorias que podem enriquecer ainda mais a experiência do usuário. Com um investimento adicional no projeto, é possível incorporar novas ferramentas e recursos, como a integração com mapas específicos para visualização detalhada dos ambientes e a ampliação das opções de conectividade, incluindo sincronização com plataformas como Google Calendar e Outlook, entre outras possibilidades técnicas.

Para futuros trabalhos de análise e desenvolvimento de sistemas voltados ao gerenciamento e controle de espaços institucionais, sugere-se uma abordagem aprofundada na análise de ferramentas e websites existentes, examinar seus processos técnicos de desenvolvimento, decisões técnicas, especialmente no que diz respeito à compatibilidade com os objetivos do projeto. Além do aprimoramento do projeto, desenvolvendo os requisitos pendentes.

REFERÊNCIAS

ATLASSIAN. **Sobre o Trello:** O que está por trás dos quadros. [S. l.], 2025. Disponível em: https://trello.com/pt-BR/about. Acesso em: 12 maio 2025.

BEEKAI. **React Hook Form**. [*S. l.*], 2025. Disponível em: https://react-hook-form.com. Acesso em: 25 mar. 2025.

CALENDA. Smartrooms: Gestão de salas descomplicada. Use nosso app de reserva de salas de reunião. Otimize o uso dos seus espaços e agende uma reunião em menos de 30 segundos. In: **Smartrooms**. [S. l.], 2025. Disponível em: https://calenda.com.br/smartrooms. Acesso em: 25 mar. 2025.

CONTRIBUTORS OF PNPM. **Pnpm**. [S. l.], 2025. Disponível em: https://pnpm.io/pt/. Acesso em: 8 abr. 2025.

DESKBEE. DESKBEE: Quebre seus recordes de produtividade gerenciando o híbrido com máxima performance. In: DESKBEE. **Deskbee.** [S. l.], 2025. Disponível em: https://br.deskbee.co/#contact. Acesso em: 25 mar. 2025.

DOTENVX. **Dotenv**. [*S. l.*], 2025. Disponível em: https://github.com/motdotla/dotenv. Acesso em: 25 mar. 2025.

ESCALA. Clicou, organizou! Oferecemos o melhor da tecnologia para otimizar seus recursos e escalar eficiência no seu negócio. In: ESCALA. **Monte escalas com agilidade e ganhe dados para ter sempre a equipe ideal** [S. l.], 2022. Disponível em: https://escala.app. Acesso em: 25 mar. 2025.

FASTIFY. **Fastify**: Fast and low overhead web framework, for Node.js. [*S. l.*], 2025. Disponível em: https://fastify.dev. Acesso em: 25 mar. 2025.

FIA. **Scrum**: o que é e como aplicar a metodologia ágil para gestão?. [S. l.], 5 fev. 2024. Disponível em: https://fia.com.br/blog/scrum/. Acesso em: 8 jun. 2025.

GITHUB. Github. *In*: **Github**. [*S. l.*], 2025. Disponível em: https://github.com. Acesso em: 25 mar. 2025.

HUSKY. Husky. *In*: HUSKY. **What is Prettier?**. [*S. l.*], 2025. Disponível em: https://typicode.github.io/husky/. Acesso em: 25 mar. 2025.

IBM. Diagramas de Caso de Uso: UML. In: IBM. Diagramas de Caso de Uso: [S. l.], 5 mar. 2021. Disponível em: https://www.ibm.com/docs/pt-br/rsm/7.5.0?topic=diagrams-use-case. Acesso em: 25 mar. 2025.

JÄPPINEN. Iiroj. Lint-staged. In: JÄPPINEN. Iiroj. **Lint-staged**. [S. l.], 2025. Disponível em: https://github.com/lint-staged/lint-staged. Acesso em: 25 mar. 2025.

JETBRAINS S.R.O. **PostgreSQL.** [S. l.], 2025. Disponível em: https://www.jetbrains.com/datagrip/features/postgresql. Acesso em: 8 abr. 2025.

LACEY, Jake. **Node-jsonwebtoken**. [*S. l.*], 2025. Disponível em: https://github.com/auth0/node-jsonwebtoken. Acesso em: 25 mar. 2025.

LEGEND80S. Git-commit-msg-linter. *In*: LEGEND80S **Git-commit-msg-linter**. [*S. l.*], 2024. Disponível em: https://github.com/legend80s/git-commit-msg-linter. Acesso em: 25 mar. 2025.

MAHAPATRA, Amitosh Swain. **Node.bcrypt.js**. [*S. l.*], 2025. Disponível em: https://github.com/kelektiv/node.bcrypt.js. Acesso em: 25 mar. 2025.

MCDONNELL, Colin. **Zod**: TypeScript-first schema validation with static type inference. [*S. l.*], 2025. Disponível em: https://zod.dev/. Acesso em: 25 mar. 2025.

META OPEN SOURSE. React The library for web and native user interfaces. *In*: META OPEN SOURSE. **React** . [*S. l.*], 2025. Disponível em: https://react.dev/. Acesso em: 25 mar. 2025.

MICROSOFT. Visual Studio Code. *In*: **Visual Studio Code**. [*S. l.*], 2025. Disponível em: https://code.visualstudio.com. Acesso em: 25 mar. 2025.

MKLABS CO.LTD. StarUML. *In*: **StarUML**. [*S. l.*], 2025. Disponível em: https://staruml.io. Acesso em: 25 mar. 2025.

OPENJS FOUNDATION. Eslint. *In*: ESLINT. **Find and fix problems in your JavaScript code**. [*S. l.*], 2025. Disponível em: https://eslint.org/. Acesso em: 25 mar. 2025.

OPENJS FOUNDATION. Node.js. *In*: **Run JavaScript Everywhere**. [*S. l.*], 2025. Disponível em: https://nodejs.org/en. Acesso em: 25 mar. 2025.

OPENJS FOUNDATION. TypeScript. *In*: MICROSOFT. **TypeScript is JavaScript with syntax for types.**. [*S. l.*], 2025. Disponível em: https://www.typescriptlang.org/. Acesso em: 25 mar. 2025.

OPENJS FOUNDATION. **Jest**. [S. l.], 2025. Disponível em: https://jestjs.io/pt-BR/. Acesso em: 8 abr. 2025.

POSTGRESQL. **About PostgreSQL**. [S. l.], 2025. Disponível em: https://www.postgresql.org/about/. Acesso em: 12 maio 2025.

PRETTIER. Prettier. *In*: PRETTIER. **What is Prettier?**. [*S. l.*], 2025. Disponível em: https://prettier.io/docs/en/. Acesso em: 25 mar. 2025.

PRISMA. **Prisma**. [*S. l.*], 2025. Disponível em: https://www.prisma.io. Acesso em: 25 mar. 2025.

REVELO. **PostCSS**: a ótima ferramenta para Frontend. [S. l.], 4 abr. 2025. Disponível em: https://community.revelo.com.br/postcss-a-otima-ferramenta-para-frontend/. Acesso em: 12 maio 2025.

SAAYMAN, Jason. **Axios**. [*S. l.*], 2025. Disponível em: https://github.com/axios/axios. Acesso em: 25 mar. 2025.

SITNIK, Andrey. **Postcss**. [*S. l.*], 2025. Disponível em: https://github.com/postcss/postcss. Acesso em: 25 mar. 2025.

SOFTWARE FREEDOM CONSERVANCY. **Git**. [*S. l.*], 2025. Disponível em: https://git-scm.com. Acesso em: 25 mar. 2025.

SOMMERVILLE, Ian. **Engenharia de software**. 10. ed. São Paulo: Pearson, 2019.

VERCEL. The React Framework for the Web. *In*: VERSEL. **Next.js**. [*S. l.*], 2025. Disponível em: https://nextjs.org/. Acesso em: 25 mar. 2025.

WATHAN, Adam. **Rapidly build modern websites without ever leaving your HTML**. [*S. l.*], 2025. Disponível em: https://tailwindcss.com. Acesso em: 25 mar. 2025.

WISER EXPERIENCE: O epicentro tecnológico de escritórios do futuro. In: WISER EXPERIENCE. **SOLUÇÕES DE WORKPLACE EXPERIENCE** [S. l.], 2023. Disponível em: https://wiserxp.com. Acesso em: 25 mar. 2025.

WORKOS. **Radix**. [*S. l.*], 2025. Disponível em: https://www.radix-ui.com. Acesso em: 25 mar. 2025.

ZHANG, Helena; FRIED, Tobias. **Phosphoricons:** Phosphor is a flexible icon family for interfaces, diagrams, presentations. [*S. l.*], 2025. Disponível em: https://phosphoricons.com. Acesso em: 25 mar. 2025.

APÊNDICE A – RECURSOS COMPLEMETARES

O Apêndice A tem como objetivo apresentar os recursos visuais complementares ao desenvolvimento do projeto, bem como indicar o repositório utilizado para armazenamento e versionamento do código-fonte. Essa seção serve como apoio visual e técnico para a compreensão completa da solução proposta.

O vídeo demonstrativo tem como finalidade apresentar, de forma prática e visual, o funcionamento da aplicação desenvolvida. Por meio da simulação de casos de uso, são exibidas as principais funcionalidades do sistema, sua interface e a interação do usuário com os recursos implementados. Esse material complementa a documentação escrita, facilitando a compreensão da dinâmica do projeto e permitindo uma avaliação mais clara de seu comportamento em tempo real. O link para o vídeo é: https://youtu.be/3Nwa29Nwk8c

O repositório do projeto foi utilizado como ferramenta central de controle de versão e organização do código-fonte, facilitando o acompanhamento das alterações realizadas durante o desenvolvimento. Ele permite o registro histórico das modificações, a colaboração entre integrantes da equipe e a rastreabilidade das implementações. Além disso, o repositório abriga toda a estrutura do projeto, incluindo arquivos de configuração, documentação técnica e demais recursos relevantes para sua execução e manutenção. O link para o repositório é: https://github.com/JulioC090/book-space/tree/v1

APÊNDICE B – METODOS DOS DIAGRAMAS DE CLASSE

O Apêndice B tem como finalidade reunir todos os quadros que descrevem os métodos utilizados nos diagramas de classe apresentados ao longo do trabalho. Essa organização visa proporcionar uma visão mais detalhada e sistemática da estrutura dos diagramas, facilitando o entendimento de sua lógica interna e do papel de cada método dentro do sistema modelado.

Quadro 28 - Métodos do AuthContext.

Método	Descrição
signIn	Método para realizar o login de um usuário.
logout	Método para realizar o logout de um usuário.

Fonte: Elaborado pelos autores (2025).

Quadro 29 - Métodos do AuthService

Método	Descrição
signUp	Método para registrar um novo usuário
signIn	Método para realizar o login de um usuário.
logout	Método para realizar o logout de um usuário.

Fonte: Elaborado pelos autores (2025).

Quadro 30 - Métodos do AxiosHttpClient

Método	Descrição
signUp	Método para registrar um novo usuário
signIn	Método para realizar o login de um usuário.
logout	Método para realizar o logout de um usuário.

Fonte: Elaborado pelos autores (2025).

Quadro 31 - Métodos do useWorkspace

Método	Descrição
addWorkspace	Adiciona uma nova *workspace*.
updateWorkspace	Atualiza um *workspace* existente.
deleteWorkspace	Deleta um *workspace*.
addUser	Adiciona um usuário a um *workspace*.
leaveWorkspace	O usuário sai da *workspace*

Fonte: Elaborado pelos autores (2025).

Quadro 32 - Métodos da WorkspaceService

Método	Descrição
loadAll	Carrega todos os *workspaces*.
load	Carrega um *workspace* específico.
add	Adiciona uma nova *workspace*.
update	Atualiza uma *workspace* existente.
delete	Deleta uma *workspace*.

Fonte: Elaborado pelos autores (2025).

Quadro 33 - Métodos da WorkspaceUserService

Método	Descrição
add	Adiciona um usuário a uma *workspace*.
update	Atualiza informações de um usuário em uma *workspace*.
delete	Remove um usuário de uma *workspace*.
leave	Usuário sai de uma *workspace*.

Fonte: Elaborado pelos autores (2025).

Quadro 34 - Métodos do useSpaces

Método	Descrição
loadSpaces	Carrega uma lista de espaços em memória e disponibiliza no estado
loadAllSpaces	Carrega todos os espaços que o usuário tem acesso
addSpace	Adiciona um novo espaço a uma *workspace*
updateSpace	Atualiza um espaço existente
deleteSpace	Deleta um espaço
bookSpace	Reserva um espaço utilizando as informações de reserva fornecidas.

Fonte: Elaborado pelos autores (2025).

Quadro 35 - Métodos do SpaceService

Método	Descrição
loadAll	Carrega todos os espaços disponíveis.
add	Adiciona um espaço
update	Atualiza um espaço existente
delete	Deleta um espaço
bookSpace	Reserva um espaço utilizando as informações de reserva fornecidas.

Fonte: Elaborado pelos autores (2025).

Quadro 36 - Métodos do useWorkspaceDetails

Método	Descrição
updateWorkspace	Atualiza os detalhes de uma *workspace*
addUser	Adiciona um usuário a *workspace*
updateUserRole	Atualiza o cargo de um usuário na *workspace*
deleteUser	Remove um usuário da *workspace*

Fonte: Elaborado pelos autores (2025).

Quadro 37 – Métodos do useWorkspaceResource

Método	Descrição
loadResources	Carrega em memória os recursos de uma *workspace*.
addResource	Adiciona um recurso a uma *workspace* específica.
deleteResource	Deleta um recurso de uma *workspace* específica.

Fonte: Elaborado pelos autores (2025).

Quadro 38 – Métodos da WorkspaceResourceService

Método	Descrição
add	Adiciona um recurso a uma *workspace* específica.
remove	Deleta um recurso de uma *workspace* específica.

Fonte: Elaborado pelos autores (2025).

Quadro 39 – Métodos do useBookings

Método	Descrição
loadAll	Carrega todas as reservas disponíveis.

Fonte: Elaborado pelos autores (2025).

Quadro 40 – Métodos do BookingsService

Método	Descrição
loadAll	Carrega todas as reservas disponíveis.

Fonte: Elaborado pelos autores (2025).

Quadro 41 – Métodos do Fastify.

Método	Descrição
post	Utilizado para criar recursos no servidor. O cliente envia dados no corpo da requisição, que o servidor processa e armazena.
addHook	Pode referir-se genericamente a adicionar rotas*, plugins* ou *hooks*.
get	Utilizado para recuperar dados do servidor. Não modifica os dados no servidor.
patch	Utilizado para atualizar parcialmente um recurso existente no servidor.
delete	Utilizado para excluir um recurso do servidor.
register	Utilizado para modularizar a aplicação, adicionando rotas, utilitários ou *hooks*.
listen	Utilizado para começar a aceitar requisições HTTP.

Fonte: Elaborado pelos autores (2025).

Quadro 42 – Método do SignUpController.

Método	Descrição
handle	Primeiro ele valida os dados recebidos na requisição usando um esquema definido com a biblioteca Zod. Se os dados não passarem na validação, uma resposta de erro 400 (*Bad Request*) é retornada, contendo informações sobre os problemas de validação.

Fonte: Elaborado pelos autores (2025).

Quadro 43 – Método do Controller.

Método	Descrição
handle	É o método responsável por receber e processar requisições https a partir de um esquema definido com a biblioteca Zod.

Fonte: Elaborado pelos autores (2025).

Quadro 44 – Método do AddAccount.

Método	Descrição
add	Recebe como parâmetro um objeto representando os dados do usuário a serem adicionados, exceto o ID, pois geralmente é gerado automaticamente pelo sistema. Retorna um valor booleano indicando se a adição foi bem-sucedida ou não.

Fonte: Elaborado pelos autores (2025).

Quadro 45 – Métodos do AccountPrismaRepository.

Método	Descrição
add	Recebe os dados de uma nova conta de usuário e os insere no banco de dados.
checkByEmail	Recebe um e-mail como entrada e verifica se esse e-mail já está associado a uma conta existente no banco de dados.
loadByEmail	
updateAccess	
loadByToken	

Fonte: Elaborado pelos autores (2025).

Quadro 46 – Métodos do BcryptAdapter.

Método	Descrição
hash	Este método recebe uma senha em texto simples e a processa usando o algoritmo de *hash Bcrypt*, retornando a senha hasheada resultante.
compare	Este método compara uma senha em texto simples com uma senha hasheada. Se a senha em texto simples corresponder à senha hasheada, ele retorna *true*, indicando uma correspondência bem-sucedida. Caso contrário, retorna *false*.

Fonte: Elaborado pelos autores (2025).

Quadro 47 – Método do IHasher.

Método	Descrição
hash	É o método responsável por calcular um valor de *hash* para os dados fornecidos.

Fonte: Elaborado pelos autores (2025).

Quadro 48 – Método do IHashComparer.

Método	Descrição
compare	É o método responsável por comparar duas *strings*, ou seja, os *hashes*, e retornar um valor *true* se a comparação for bem-sucedida. Caso contrário, retorna false.

Fonte: Elaborado pelos autores (2025).

Quadro 49 – Método do SignInController.

Método	Descrição
handle	Implementa o método definido na interface *Controller*. Este método é chamado quando uma requisição HTTP é recebida e é responsável por processar essa requisição.

Fonte: Elaborado pelos autores (2025).

Quadro 50 – Método do Authentication.

Método	Descrição
auth	É o método responsável por realizar o processo de autenticação de usuário a partir de uma entrada *AuthenticationInput* do tipo *user*.

Fonte: Elaborado pelos autores (2025).

Quadro 51 – Método do JWTAdapter.

Método	Descrição
encrypt	Implementa o método definido na interface *IEncrypter*, que é responsável por criptografar dados. Este método recebe um texto simples (*plaintext*) como entrada e retorna uma *string* representando o token JWT criptografado.

Fonte: Elaborado pelos autores (2025).

Quadro 52 – Método do IEncrypter.

Método	Descrição
encrypt	É o método responsável por cifrar e decifrar os dados de entrada.

Fonte: Elaborado pelos autores (2025).

Quadro 53 – Método do AuthMiddleware.

Método	Descrição
handle	Implementa o método definido na interface *Middleware*, que é chamado quando uma requisição HTTP é interceptada pelo middleware de autenticação.

Fonte: Elaborado pelos autores (2025).

Quadro 54 – Método do Middleware.

Método	Descrição
handle	Este método define a assinatura que um middleware deve implementar. Ele recebe um objeto *reques*t representando a requisição HTTP e retorna uma promessa (*Promise*) que resolve para um objeto response representando a resposta HTTP. O método *handle* é responsável por processar a requisição e produzir uma resposta.

Fonte: Elaborado pelos autores (2025).

Quadro 55 – Método do LogoutController.

Método	Descrição
handle	Implementa o método definido na interface *Controller*. Este método é chamado quando uma requisição HTTP é recebida para realizar o logout de um usuário.

Fonte: Elaborado pelos autores (2025).

Quadro 56 – Método do Logout.

Método	Descrição
logout	Este método recebe o ID da conta do usuário que deseja fazer logout e executa a lógica de logout.

Fonte: Elaborado pelos autores (2025).

Quadro 57 – Método do DeleteLeaveWorkspaceController.

Método	Descrição
handle	É responsável por processar uma requisição para que um usuário saia de um espaço de trabalho. Ele recebe os dados da requisição, chama o caso de uso apropriado para executar a lógica de negócio necessária e retorna uma resposta HTTP indicando o resultado da operação.

Fonte: Elaborado pelos autores (2025).

Quadro 58 – Método do LeaveWorkspace.

Método	Descrição
leave	Possibilita que um usuário saia de um espaço de trabalho. Primeiro, ele verifica o papel do usuário nesse espaço. Em seguida, confere se esse papel tem permissão para realizar a saída. Se sim, o usuário é removido do espaço de trabalho. Finalmente, o método retorna um indicativo de sucesso ou falha, informando se a operação foi concluída com êxito.

Fonte: Elaborado pelos autores (2025).

Quadro 59 – Método do WorkspacePrismaRepository.

Método	Descrição
checkUserInWorkspace	Este método verifica se um usuário está associado a um determinado *workspace*, Ele realiza uma consulta ao banco de dados procurando pelo *workspace* com o ID especificado, se o *workspace* for encontrado e o usuário for o proprietário (*owner*), retorna *true*, caso contrário, verifica se o usuário está associado ao *workspace* como membro e retorna *true* se encontrar ou false caso contrário.
loadById	Este método carrega os detalhes de um *workspace* com base no seu ID.
deleteUserInWorkspace	Este método exclui um usuário de um determinado *workspace*. Realiza uma operação de exclusão no banco de dados na tabela
load	Carrega dados do usuário.
add	Adiciona um novo *workspace*.
update	Atualiza dados do *workspace*.
delete	Deleta um *workspace*.
addUserToWorkspace	Adiciona um usuário ao *workspace*.
loadWorkspaceDetails	Carrega detalhes do *workspace*.
updateUserRole	Atualiza o papel de um usuário no *workspace*.
loadUserRole	Carrega o papel de um usuário no *workspace*.

Fonte: Elaborado pelos autores (2025).

Quadro 60 – Método do DeleteUserInWorkspaceController.

Método	Descrição
handle	É responsável por lidar com as requisições HTTP relacionadas à exclusão de um usuário de um *workspace*. Ele recebe uma requisição HTTP, extrai os dados necessários, como o ID do usuário e do *workspace*, e os repassa ao caso de uso *DeleteUserInWorkspace*. Em seguida, ele retorna uma resposta HTTP adequada com base no resultado da operação de exclusão.

Fonte: Elaborado pelos autores (2025).

Quadro 61 – Método do DeleteUserInWorkspace.

Método	Descrição
deleteUserInWorkspace	É responsável por coordenar a remoção de um usuário específico de um *workspace*. Ele recebe como entrada o usuário autenticado, o ID do *workspace* e o e-mail do usuário a ser removido.

Fonte: Elaborado pelos autores (2025).

Quadro 62 – Método do DeleteWorkspaceController.

Método	Descrição
handle	Primeira etapa é validar os parâmetros da solicitação, garantindo que o *workspaceId* seja uma *string* válida. Se a validação falhar, retorna uma resposta de erro *badRequest* com os detalhes do erro de validação. Em seguida, o controlador extrai o ID da conta do usuário autenticado da solicitação. Em seguida, utiliza o serviço *DeleteWorkspace* para tentar excluir o *workspace.* Ele passa o ID da conta e o ID do *workspace* para o método delete do serviço.

Fonte: Elaborado pelos autores (2025).

Quadro 63 – Método do DeleteWorkspace.

Método	Descrição
delete	Verifica se o usuário autenticado tem permissão para excluir o *workspace*. Se tiver permissão, o método utiliza o repositório correspondente para excluir o *workspace* e retorna *true*. Caso contrário, retorna *false*.

Fonte: Elaborado pelos autores (2025).

Quadro 64 – Método do GetWorkspaceDetailsController.

Método	Descrição
handle	Recebe a requisição HTTP, que inclui o ID do espaço de trabalho desejado nos parâmetros. Ele valida os parâmetros da requisição e, em seguida, utiliza a instância de *LoadWorkspaceDetails* para carregar os detalhes do espaço de trabalho com base na ID fornecido.

Fonte: Elaborado pelos autores (2025).

Quadro 65 – Método do LoadWorkspaceDetails.

Método	Descrição
loadWoekspaceDetails	Carrega os detalhes de um espaço de trabalho com base no ID fornecido, incluindo informações como nome, descrição, membros e permissões associadas. Ele verifica se o usuário autenticado está associado ao espaço de trabalho, carrega os detalhes do espaço de trabalho e o papel do usuário dentro desse espaço de trabalho, e retorna essas informações encapsuladas em um objeto. Se o usuário não estiver associado ao espaço de trabalho ou se os detalhes não puderem ser carregados, o método retorna *null*.

Fonte: Elaborado pelos autores (2025).

Quadro 66 – Método do GetWorkspaceController.

Método	Descrição
handle	Ele extrai o ID da conta do usuário da requisição e utiliza o caso de uso *LoadWorkspaces* para carregar os espaços de trabalho associados a esse usuário. Em seguida, retorna uma resposta HTTP com o código `200 OK` e os espaços de trabalho encontrados no corpo da resposta.

Fonte: Elaborado pelos autores (2025).

Quadro 67 – Método do LoadWorkspace.

Método	Descrição
load	Tem como objetivo carregar os espaços de trabalho associados a um usuário específico. Ele recebe como entrada o ID do usuário e utiliza um repositório específico para buscar os espaços de trabalho correspondentes. Após realizar a consulta, retorna uma lista contendo os objetos *Workspace* que representam os espaços de trabalho encontrados. Se não houver espaços de trabalho associados ao usuário ou se ocorrer algum problema durante o processo de carregamento, o método pode retornar uma lista vazia.

Fonte: Elaborado pelos autores (2025).

Quadro 68 – Método do PatchWorkspaceController.

Método	Descrição
handle	Este método é chamado para lidar com a requisição PATCH. Ele primeiro valida os parâmetros e o corpo da requisição. Em seguida, verifica se o corpo da requisição não está vazio. Depois, extrai o ID do usuário autenticado da requisição. Em seguida, chama o caso de uso *updateWorkspace*, passando os dados validados e o ID do usuário autenticado. Por fim, retorna uma resposta HTTP adequada com base no resultado da atualização, sendo `200 OK` se a atualização for bem-sucedida e `400 BadRequest` se ocorrer algum problema.

Fonte: Elaborado pelos autores (2025).

Quadro 69 – Método do UpdateWorkspace.

Método	Descrição
update	Este método é responsável por realizar a atualização do espaço de trabalho. Primeiro, verifica se o usuário autenticado possui permissões adequadas para atualizar o espaço de trabalho. Isso é feito consultando o papel (role) do usuário no espaço de trabalho por meio do *loadUserRoleRepository.* Em seguida, verifica se o papel do usuário possui permissões para atualizar o espaço de trabalho com base nas *WorkspacePermissions*. Se o usuário tiver permissões adequadas, chama o método update do *IUpdateWorkspaceRepository* para realizar a atualização. Retorna *true* se a atualização for bem-sucedida e false caso contrário.

Fonte: Elaborado pelos autores (2025).

Quadro 70 – Método do PostUserToWorkspaceController.

Método	Descrição
handle	É responsável por lidar com a requisição HTTP de adição de usuário a um espaço de trabalho.

Fonte: Elaborado pelos autores (2025).

Quadro 71 – Método do AddUserToWorkspace.

Método	Descrição
addUserToWorkspace	Este método é responsável por executar a adição de um usuário a um espaço de trabalho. Ele recebe como parâmetros o usuário autenticado que está realizando a adição, o ID do espaço de trabalho e as informações do usuário a ser adicionado (como o e-mail e o papel que terá no espaço de trabalho). Este método coordena várias etapas, incluindo a verificação de permissões, a validação dos parâmetros e a interação com os repositórios de dados para efetuar a adição do usuário ao espaço de trabalho.

Fonte: Elaborado pelos autores (2025).

Quadro 72 – Método do PostWorkspaceController.

Método	Descrição
handle	Válida a requisição, extrai o ID da conta do usuário autenticado, chama o caso de uso responsável por criar o espaço de trabalho e retorna uma resposta adequada com base no resultado da operação.

Fonte: Elaborado pelos autores (2025).

Quadro 73 – Método do AddWorkspace.

Método	Descrição
addUserToWorkspace	Tem como objetivo adicionar um usuário a um espaço de trabalho, verificando se o usuário autenticado tem permissão para realizar essa ação e se o usuário a ser adicionado já não está presente no espaço de trabalho

Fonte: Elaborado pelos autores (2025).

Quadro 74 – Método do PostWorkspaceUserRoleController.

Método	Descrição
handle	É o método responsável por processar as requisições HTTP relacionadas a definição ou alteração das funções ou cargos dos usuários na *workspace*.

Fonte: Elaborado pelos autores (2025).

Quadro 75 – Método do UpdateWorkspaceUserRole.

Método	Descrição
updateUserRole	É o método responsável por realizar a alteração da função ou cargo de um usuário em uma *workspace*. O método realiza a autenticação do usuário que irá realizar a alteração e recupera as informações da função do usuário alterado para atualizá-la naquela *workspace*.

Fonte: Elaborado pelos autores (2025).

Quadro 76 – Método do PostWorkspaceResourceController.

Método	Descrição
handle	Manipula a requisição HTTP.

Fonte: Elaborado pelos autores (2025).

Quadro 77 – Método do AddWorkspaceResource.

Método	Descrição
add	Adiciona um recurso ao *workspace* autenticado.

Fonte: Elaborado pelos autores (2025).

Quadro 78 – Método do WorkspaceResourcePrismaRepository

Método	Descrição
add	Adiciona um recurso ao *workspace* autenticado.
delete	Deleta um recurso do *workspace*.
checkName	Verifica se o nome de um recurso já existe no *workspace*.

Fonte: Elaborado pelos autores (2025).

Quadro 79 – Método do PostSpaceController.

Método	Descrição
handle	Manipula a requisição HTTP para criação de um novo espaço.

Fonte: Elaborado pelos autores (2025).

Quadro 80 – Métodos do AddSpace.

Método	Descrição
add	Adiciona um novo espaço ao *workspace* autenticado com os recursos opcionais especificados.

Fonte: Elaborado pelos autores (2025).

Quadro 81 – Métodos do SpacePrismaRepository.

Método	Descrição
isInWorkspace	Verifica se um espaço está dentro de um *workspace*.
verifyUserAccess	Verifica se um usuário tem acesso a um espaço.
loadSpaceAvailability	Carrega a disponibilidade de um espaço.
loadUserSpaces	Carrega os espaços de um usuário.
add	Adiciona um espaço a um *workspace*.
update	Atualiza um espaço.
delete	Deleta um espaço.

Fonte: Elaborado pelos autores (2025).

Quadro 82 – Método do PostBookController.

Método	Descrição
handle	Método que manipula a requisição.

Fonte: Elaborado pelos autores (2025).

Quadro 83 – Método do AddBook.

Método	Descrição
add	Método para adicionar uma reserva.

Fonte: Elaborado pelos autores (2025).

Quadro 84 – Métodos do BookingPrismaRepository

Método	Descrição
loadAll	Carrega todas as reservas de uma conta.
loadBookingsByDay	arrega reservas por dia para um espaço específico.
addBooking	Adiciona uma nova reserva.

Fonte: Elaborado pelos autores (2025).

Quadro 85 – Método do PatchSpaceController.

Método	Descrição
handle	Método que manipula a requisição.

Fonte: Elaborado pelos autores (2025).

Quadro 86 – Método do UpdateSpace.

Método	Descrição
update	Método para atualizar um espaço.

Fonte: Elaborado pelos autores (2025).

Quadro 87 – Método do GetSpacesController.

Método	Descrição
handle	Método que manipula a requisição.

Fonte: Elaborado pelos autores (2025).

Quadro 88 – Método do LoadSpaces.

Método	Descrição
load	Método para carregar espaços.

Fonte: Elaborado pelos autores (2025).

Quadro 89 – Método do GetSpaceBookingsController.

Método	Descrição
handle	Método que manipula a requisição.

Fonte: Elaborado pelos autores (2025).

Quadro 90 – Método do LoadSpaceBookings.

Método	Descrição
load	Método para carregar as reservas de um espaço para um usuário autenticado em um determinado dia.

Fonte: Elaborado pelos autores (2025).

Quadro 91 – Método do GetBookingsController.

Método	Descrição
handle	Método que manipula a requisição.

Fonte: Elaborado pelos autores (2025).

Quadro 92 – Método do LoadBookings.

Método	Descrição
loadall	Método para carregar todas as reservas para um usuário autenticado.

Fonte: Elaborado pelos autores (2025).

Quadro 93 – Método do DeleteWorkspaceResourceController.

Método	Descrição
handle	Controlador genérico para processar requisições HTTP.

Fonte: Elaborado pelos autores (2025).

Quadro 94 – Método do DeleteWorkspaceResource.

Método	Descrição
delete	Método para deletar um recurso específico de um *workspace* para um usuário autenticado.

Fonte: Elaborado pelos autores (2025).

Quadro 95 – Método do DeleteSpaceController.

Método	Descrição
handle	Este método é responsável por processar a requisição HTTP recebida. Quando uma requisição de exclusão de espaço é recebida, este método será chamado. Ele provavelmente extrai informações da requisição (como o ID do usuário autenticado, o ID do *workspace* e o ID do espaço) e delega a operação de exclusão ao componente apropriado (possivelmente `*DeleteSpace*`).

Fonte: Elaborado pelos autores (2025).

Quadro 96 – Método do DeleteSpace.

Método	Descrição
delete	Este método realiza a operação de exclusão do espaço.

Fonte: Elaborado pelos autores (2025).