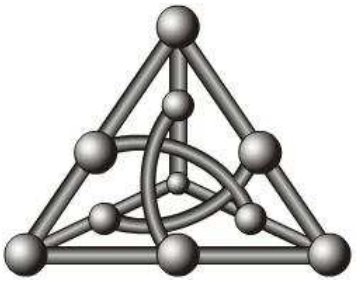
Desenvolvimento de Sistema Web para Agendamento de Espaços Físicos do HUMAP/UFMS

FACOM
Faculdade de Computação
Universidade Federal de Mato Grosso do Sul



Autor: Wellington Evangelista Idino

Orientadora: Profa. Dra. Liana Dessandre Duenha Garanhani

Trabalho de Conclusão de Curso.

04 de dezembro de 2025

Sumário

Lista de Figuras **iii**

Lista de Abreviaturas e Siglas **iv**

1 Introdução **1**

1.1 Objetivos 2

1.1.1 Objetivos Específicos 2

1.2 Justificativa 3

1.3 Organização do Trabalho 3

2 Referencial Teórico **4**

2.1 Engenharia de *software* e Metodologias de Desenvolvimento 4

2.2 Arquitetura de Sistemas *Web* 5

2.3 Containerização e Infraestrutura 5

2.4 Segurança em Sistemas *Web* 5

2.5 Testes e Validação de *software* 6

3 Metodologia **8**

3.1 Processo de Desenvolvimento 8

3.2 Validação e Testes 9

3.3 Planejamento e Arquitetura da Solução 10

3.4 Descrição Técnica da Arquitetura 10

3.4.1 Principais Módulos do *Frontend* 12

3.4.2 Principais Módulos do *Backend* 12
3.4.3 Estrutura do Banco de Dados 13
3.4.4 Containerização e Infraestrutura 15

4 Resultados **16**
4.1 Visão Geral do Sistema 16
4.2 Módulo de Autenticação e Gestão de Usuários 17
4.3 Gerenciamento de Reservas e Interface de Calendário 17
4.4 Painel Administrativo 18
4.5 Entrega Técnica e Prontidão do Sistema 18

5 Considerações Finais **28**
5.1 Contribuições 28
5.2 Limitações e Trabalhos Futuros 28
5.3 Considerações Finais 29

6 Referências Bibliográficas **30**

A Documentação do Sistema **32**

Lista de Figuras

3.1 Arquitetura Geral do Sistema HUBook 11

3.2 Diagrama Entidade-Relacionamento (DER) Conceitual 14

4.1 Página Inicial do Sistema HUBook 20

4.2 Fluxo de Criação e Aprovação de Reserva 21

4.3 Página de *Login* 22

4.4 Página de Registro de Usuário 22

4.5 Página de Perfil do Usuário 23

4.6 Página do Calendário de Reservas 23

4.7 Página do Formulário de Criação de Reserva 24

4.8 Página Minhas Solicitações 25

4.9 Diagrama de Entidades-Relacionamentos Físico 26

4.10 Página do Painel Administrativo - Gerenciamento de Reservas 27

4.11 Página do Painel Administrativo - Gerenciamento de Salas 27

Lista de Abreviaturas e Siglas

API	*Application Programming Interface* (Interface de Programação de Aplicações)
CPF	Cadastro de Pessoa Física
CRUD	*Create, Read, Update, Delete* (Criar, Ler, Atualizar, Deletar)
CSRF	*Cross-Site Request Forgery* (Falsificação de Solicitação Entre Sites)
DER	Diagrama Entidade-Relacionamento
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
HUMAP	Hospital Universitário Maria Aparecida Pedrossian
MVC	*Model-View-Controller* (Modelo-Visualização-Controlador)
MySQL	*My Structured Query Language* (Linguagem de Consulta Estruturada)
Nginx	*Engine X* (Servidor web de código aberto)
PHP	*PHP: Hypertext Preprocessor*
PHP-FPM	*PHP FastCGI Process Manager* (Gerenciador de Processos FastCGI do PHP)
PWA	*Progressive Web App* (Aplicativo Web Progressivo)
REST	*Representational State Transfer* (Transferência de Estado Representacional)
SPA	*Single Page Application* (Aplicação de Página Única)
SQL	*Structured Query Language* (Linguagem de Consulta Estruturada)
TI	Tecnologia da Informação
UFMS	Universidade Federal de Mato Grosso do Sul
XP	*Extreme Programming* (Programação Extrema)
XSS	*Cross-Site Scripting* (Script Entre Sites)

Capítulo 1

Introdução

O Hospital Universitário Maria Aparecida Pedrossian (HUMAP)/Universidade Federal de Mato Grosso do Sul (UFMS) desempenha papel fundamental na integração entre ensino, pesquisa e assistência à saúde. Em sua estrutura, diversos espaços físicos, como salas de reunião, auditórios e laboratórios, são frequentemente utilizados para atividades acadêmicas, administrativas e de extensão. No entanto, o processo de agendamento desses espaços é realizado de forma descentralizada, através de diferentes canais de comunicação, incluindo e-mail, WhatsApp, formulários online e calendários compartilhados, gerando sobrecarga administrativa, conflitos de horários e transparência de disponibilidade.

A ausência de um sistema informatizado e centralizado para o gerenciamento de reservas de espaços físicos do HUMAP/UFMS resulta em limitações que impactam diretamente a eficiência operacional do hospital, a satisfação dos usuários e a capacidade de gestão estratégica dos recursos físicos disponíveis. Dentre as limitações, pode-se citar as principais:

- **Conflitos de agenda**: Reservas para o mesmo espaço no mesmo horário, identificadas posteriormente;
- **Retrabalho administrativo**: Gerenciamento de solicitações através de múltiplos canais, aumentando carga de trabalho e possibilidade de erros;
- **Falta de transparência**: Ausência de visibilidade clara sobre disponibilidade dos espaços e status das solicitações;
- **Dificuldade de acompanhamento**: Ausência de histórico centralizado dificulta análise de utilização e planejamento;
- **Responsabilização pelo uso:** Ausência de co-responsabilização pelo uso dos espaços, gerando dificuldades de gestão adequada do patrimônio público; e
- **Falta de padronização**: Diferentes setores utilizam processos distintos, dificultando uniformização e acompanhamento.

Diante desse contexto, tornou-se necessário desenvolver uma solução tecnológica para centralizar, padronizar e democratizar o processo de reserva de espaços físicos do HU-

MAP/UFMS. A demanda por esse sistema foi apresentada pela equipe de Gestão do HUMAP/UFMS, e seu desenvolvimento ocorreu no contexto do Programa de Iniciação Tecnológica, fomentado pelos editais Nº 08/2022 e Nº 08/2024 HUMAP/UFMS.

1.1 Objetivos

O objetivo central do projeto é desenvolver um sistema *web* para o agendamento de espaços físicos do HUMAP/UFMS, denominado HUBook, que permita a solicitação, aprovação e acompanhamento de reservas de forma padronizada, eficiente e transparente.

1.1.1 Objetivos Específicos

Com base nos requisitos funcionais e não funcionais, os objetivos específicos deste trabalho são:

1. Implementar sistema cadastro e autenticação (*login* por e-mail ou Cadastro de Pessoa Física (CPF), recuperação de senha);
2. Desenvolver módulo de gerenciamento de espaços físicos (setores, tipos de salas e salas);
3. Implementar sistema de solicitação de reservas com validação automática de capacidade e verificação de conflitos de horário;
4. Desenvolver painel administrativo para análise, aprovação e rejeição de solicitações;
5. Implementar interface para visualização de calendário público e privado com filtros;
6. Desenvolver área de acompanhamento pessoal de reservas (edição e cancelamento);
7. Garantir usabilidade através de interface responsiva e intuitiva;
8. Implementar arquitetura desacoplada e containerizada, facilitando implantação e manutenção;
9. Documentar o sistema através de documentação técnica, casos de uso e modelagem; e
10. Realizar testes automatizados e testes exploratórios, complementados por demonstrações para coleta de feedback.

A implantação do sistema não fez parte dos objetivos, visto que depende da autorização da gestão do hospital e da disponibilidade da equipe de Tecnologia da Informação do HUMAP/UFMS, que deverá dar acompanhamento e manutenção no sistema a partir da sua implantação e uso.

1.2 Justificativa

A implementação de um sistema centralizado de reservas traz **impactos** significativos para o HUMAP/UFMS em diferentes frentes:

- **Operacionais:** Redução da sobrecarga administrativa através da automatização de validações e verificações; eliminação de conflitos de horário por meio de verificação automática de disponibilidade; padronização do processo de solicitação e aprovação em todos os setores; e redução do retrabalho causado pela fragmentação de canais de comunicação;
- **Ensino:** Facilita o planejamento e execução de atividades acadêmicas, permitindo que docentes, servidores técnicos, residentes e discentes visualizem a disponibilidade de espaços de forma transparente e realizem reservas de maneira ágil e confiável;
- **Pesquisa:** Organiza e otimiza o agendamento de laboratórios e espaços destinados a atividades de pesquisa, permitindo melhor aproveitamento desses recursos estratégicos;
- **Gestão:** Proporciona visibilidade completa sobre a utilização dos espaços físicos através de histórico centralizado, possibilitando análises sobre padrões de uso, identificação de espaços subutilizados ou sobrecarregados, controle e acompanhamento do bom uso do patrimônio público e embasamento para decisões estratégicas sobre infraestrutura.

Além disso, o desenvolvimento deste sistema representa uma contribuição prática para a modernização dos processos administrativos de reservas de espaços físicos do hospital e melhoria dos serviços prestados à comunidade.

1.3 Organização do Trabalho

Este trabalho está organizado em cinco capítulos principais. O Capítulo 2 apresenta o referencial teórico sobre engenharia de software, metodologias ágeis e arquitetura de sistemas *web*. O Capítulo 3 descreve a metodologia adotada no desenvolvimento. O Capítulo 4 apresenta os resultados obtidos, detalhando requisitos implementados e funcionalidades desenvolvidas. O Capítulo 5 apresenta as considerações finais. Complementando o trabalho, o apêndice A, apresenta a documentação detalhada de requisitos e o resumo de casos de usos.

Capítulo 2

Referencial Teórico

Este capítulo reúne os principais conceitos que fundamentam o desenvolvimento do sistema de agendamento de espaços físicos, o HUBook. São apresentados tópicos relacionados à *engenharia de software*, às metodologias utilizadas no processo de construção da aplicação e aos elementos arquiteturais característicos de sistemas *web*, além de aspectos de segurança e testes necessários para garantir a qualidade de um sistema desse tipo.

2.1 Engenharia de *software* e Metodologias de Desenvolvimento

A engenharia de *software* pode ser entendida como um conjunto de práticas, princípios e métodos que visam organizar e orientar o desenvolvimento de sistemas, de forma que o produto final seja confiável e adequado às necessidades dos usuários [15]. Conforme discute Sommerville [20], esse processo envolve etapas que começam no levantamento das necessidades e se estendem até a manutenção do *software* após sua implantação, contemplando análise, projeto, codificação, integração e testes.

No levantamento de requisitos, costuma-se diferenciar dois tipos principais: os *funcionais*, que descrevem o que o sistema deve realizar, e os *não funcionais*, associados a qualidades esperadas do produto, como desempenho, segurança e facilidade de uso [15]. Uma técnica amplamente adotada para representar comportamentos esperados do sistema são os casos de uso, que descrevem interações entre os usuários e o *software* de forma estruturada.

Em relação às metodologias de desenvolvimento, práticas ágeis ganharam espaço por favorecerem ciclos curtos de entrega, revisão contínua e adaptação rápida a mudanças. O *Extreme Programming* (XP), por exemplo, enfatiza a comunicação com o cliente, refatorações frequentes e validação constante do código [1]. Para Sommerville [20], esse tipo de abordagem reduz riscos, uma vez que funcionalidades vão sendo incorporadas gradualmente e podem ser avaliadas a cada iteração. Outro ponto importante é o uso de protótipos, que permite ao usuário visualizar parte do sistema antes da implementação definitiva e ajustar sua percepção sobre a solução desejada [15].

2.2 Arquitetura de Sistemas *Web*

Grande parte das aplicações atuais utiliza a arquitetura cliente-servidor, na qual o navegador (cliente) é responsável pela interface e pela interação com o usuário, enquanto o servidor concentra a lógica de negócio e o acesso ao banco de dados [15]. Essa separação facilita a manutenção, torna a aplicação mais escalável e permite que diferentes equipes trabalhem paralelamente.

Entre os padrões arquiteturais mais conhecidos está o *Model–View–Controller* (MVC) [8], presente em diversos *frameworks* modernos. No MVC, a lógica de negócio fica organizada no modelo, enquanto a visualização e a interação com o usuário são tratadas por componentes de interface; o controlador faz a mediação entre esses dois elementos [20]. O *framework Laravel*, utilizado no *backend* do HUBook, segue esse padrão [8].

Outro conceito amplamente difundido são as *Application Programming Interfaces* (API) *Representational State Transfer* (REST), que estabelecem uma forma padronizada de troca de informações entre sistemas por meio do protocolo *Hypertext Transfer Protocol* (HTTP). A adoção desse modelo permite que diferentes tecnologias consumam os mesmos recursos, como aplicações *web*, aplicativos móveis ou outros sistemas corporativos. Em paralelo, é comum o uso de *Single Page Applications* (SPA), nas quais a maior parte da lógica de interface acontece no navegador, proporcionando transições mais rápidas e uma experiência mais fluida para o usuário. Tecnologias como *React* [16] e *TypeScript* [7] têm sido amplamente utilizadas nesse contexto.

2.3 Containerização e Infraestrutura

A consolidação de ferramentas como *Docker* favoreceu um novo modo de organizar ambientes de desenvolvimento. Em vez de configurar manualmente servidores e dependências, é possível empacotar aplicações e seus componentes em *containers*, que funcionam de forma isolada e previsível [3]. O uso do *Docker Compose* amplia essa vantagem ao permitir que múltiplos serviços (como servidor *web*, banco de dados e aplicações) sejam executados conjuntamente, mantendo uma estrutura organizada e reprodutível [4].

Esse tipo de abordagem reduz problemas comuns na implantação de sistemas, como divergências entre ambientes de desenvolvimento e produção. Além disso, facilita a escalabilidade, já que serviços podem ser replicados de forma independente conforme a demanda.

2.4 Segurança em Sistemas *Web*

A segurança tem papel central em sistemas que tratam dados sensíveis, como informações pessoais ou registros institucionais. No contexto *web*, dois mecanismos básicos são frequentemente mencionados: autenticação, que identifica o usuário, e autorização, que define o que cada pessoa pode fazer dentro do sistema [15]. O *Laravel Sanctum*, adotado no HUBook, provê um modelo simples de autenticação baseada em sessões ou *tokens*, voltado

tanto para aplicações monolíticas quanto para SPAs [9].

Aplicações *web* também estão sujeitas a diferentes tipos de ataques, como *Structured Query Language (SQL) Injection*, *Cross-Site Scripting (XSS)* e *Cross-Site Request Forgery (CSRF)*. A prevenção desses problemas normalmente envolve validação rigorosa de dados de entrada, sanitização, uso de *prepared statements* e mecanismos automáticos oferecidos pelo *framework* [20]. Da mesma forma, boas práticas como criptografia de senhas, tratamento adequado de erros e ocultação de informações sensíveis contribuem para reduzir riscos.

2.5 Testes e Validação de *software*

A atividade de teste é um dos pilares fundamentais para assegurar a qualidade de um sistema. Na literatura clássica [11], destaca que o objetivo principal dos testes não é demonstrar que o *software* funciona corretamente, mas revelar comportamentos incorretos e situações que possam comprometer sua confiabilidade. De modo geral, o processo de teste verifica se o *software* atende às expectativas expressas nos requisitos e se apresenta comportamento consistente em diferentes cenários.

Segundo Sommerville [20], os testes podem ser organizados em diferentes níveis, cada um com foco específico. Os ***testes unitários*** concentram-se na menor parte testável do sistema, como funções, métodos ou componentes isolados, permitindo identificar falhas localizadas e facilitando a depuração. Em seguida, os ***testes de integração*** avaliam como módulos ou serviços distintos interagem entre si, garantindo que o comportamento esperado seja mantido quando componentes passam a trabalhar em conjunto.

Pressman [15] complementa que testes funcionais, também chamados de ***testes de caixa-preta***, avaliam o *software* a partir da perspectiva do usuário, verificando se as entradas fornecidas produzem as saídas esperadas, sem considerar a estrutura interna do código. Nesse tipo de teste, o foco está na verificação do comportamento externo da aplicação, o que é particularmente relevante em sistemas baseados em interface gráfica ou serviços acessados por múltiplos perfis de usuário.

Outro nível frequentemente mencionado é o ***teste de aceitação***, voltado à validação final do sistema pelo cliente ou pelos usuários reais. Myers. [11] definem esse teste como a etapa na qual o *software* é avaliado em relação aos requisitos de negócio, verificando se a solução desenvolvida atende às necessidades operacionais do ambiente ao qual se destina. Diferentemente dos testes técnicos, o teste de aceitação considera o ponto de vista institucional, incluindo fluxo de trabalho, aderência ao processo e adequação às rotinas de uso.

Ferramentas de automação, como o *PHPUnit* [2], têm se mostrado fundamentais para ampliar a confiabilidade do processo de verificação. A automação permite a repetição sistemática de testes e facilita a detecção precoce de regressões durante a evolução do código. Em arquiteturas que utilizam APIs, os testes de integração assumem papel ainda mais relevante, pois garantem a coerência das regras de negócio e a estabilidade das interfaces entre componentes [20, 15].

De forma geral, a validação de *software* representa o conjunto de atividades voltadas

a confirmar que o sistema final está alinhado aos requisitos inicialmente especificados, enquanto os testes verificam o comportamento técnico de suas partes. A combinação desses mecanismos contribui para a construção de aplicações mais seguras, estáveis e adequadas ao contexto de uso.

Capítulo 3

Metodologia

Este capítulo descreve a metodologia adotada no desenvolvimento do HUBook, abrangendo o planejamento arquitetural, o processo de desenvolvimento e os procedimentos de teste e validação aplicados ao sistema.

A demanda do software foi apresentada pela equipe de Gestão do HUMAP/UFMS e foi desenvolvida no contexto do Programa de Iniciação Tecnológica, proposto e fomentado pelos editais Nº08/2022 e Nº08/2024 HUMAP/UFMS.

O trabalho caracteriza-se como pesquisa aplicada, de natureza tecnológica, orientada à criação de um artefato de software capaz de solucionar um problema real de gestão de espaços físicos institucional [15]. A proposta visa não apenas demonstrar a viabilidade técnica do sistema, mas também oferecer uma solução prática ao fluxo de agendamentos do HUMAP/UFMS, integrando conceitos fundamentais da Engenharia de Software com necessidades operacionais da instituição.

3.1 Processo de Desenvolvimento

O desenvolvimento foi conduzido através de uma abordagem iterativa e incremental baseada nos princípios do *Extreme Programming* (XP) [1]. Diferente de modelos lineares rígidos, o processo adotado caracterizou-se pela flexibilidade e pela retroalimentação constante: o avanço entre as fases de concepção, design e codificação não foi unidirecional, permitindo o retorno a etapas anteriores para refinamentos, correções ou expansão de requisitos sempre que a validação prática indicava essa necessidade.

O ciclo de vida do projeto evoluiu através da interdependência entre as seguintes atividades:

Levantamento e Evolução dos Requisitos. A identificação do problema da descentralização dos agendamentos marcou o início do projeto, mas a definição dos requisitos manteve-se dinâmica. Embora as reuniões iniciais com a Unidade de Gestão de graduação, Ensino Técnico e Extensão e o setor de Tecnologia da Informação (TI) tenham estabelecido a base funcional, o escopo foi revisitado diversas vezes. À medida que o desenvolvimento

avançava e novas nuances operacionais surgiam, a equipe retornava à etapa de análise para redefinir regras de negócio, garantindo que o *software* permanecesse alinhado às necessidades reais.

Validação por Prototipagem. A prototipagem não serviu apenas como etapa de design, mas como ferramenta de validação de fluxo. A apresentação de modelos de baixa fidelidade aos setores envolvidos permitiu antecipar gargalos lógicos antes da escrita do código. Os *feedbacks* recebidos nesta fase frequentemente exigiram o retorno à definição dos requisitos para ajustes no fluxo de aprovação, demonstrando como a validação visual precoce evitou o retrabalho técnico posterior.

Construção Incremental e Adaptação Técnica. A implementação do sistema (envolvendo *frontend* em React e *backend* em Laravel) ocorreu em ciclos de construção e adaptação. Um exemplo claro dessa não-linearidade foi o tratamento da autenticação: ao identificar a inviabilidade técnica da integração institucional durante a fase de codificação, foi necessário retroceder ao planejamento da arquitetura. Esse movimento permitiu incorporar um novo módulo de gestão de identidade (Laravel Sanctum [9]), ajustando o escopo técnico sem comprometer a continuidade do projeto.

Integração e Refinamento Recorrente. A unificação entre interface e lógica de banco de dados não representou o fim do desenvolvimento, mas um novo ciclo de refinamento. A visualização do sistema integrado revelou necessidades de usabilidade que não eram perceptíveis nas etapas isoladas. Isso demandou o retorno à camada de *frontend* para a implementação de melhorias, como filtros de pesquisa mais robustos e ajustes visuais no painel administrativo.

3.2 Validação e Testes

Os testes foram realizados no ambiente local de desenvolvimento e concentraram-se na verificação interna do sistema, uma vez que o HUMAP/UFMS não disponibilizou ambiente oficial de homologação para execução de testes por usuários finais. Assim, a validação envolveu testes automatizados e testes exploratórios conduzidos pelo desenvolvedor.

Testes Unitários. Utilizando *PHPUnit* [2], foram implementados testes destinados a verificar partes isoladas do sistema, especialmente trechos de lógica sensível, validações internas e comportamentos de componentes específicos.

Testes de Integração. Também com *PHPUnit*, foram elaborados testes de integração para verificar o funcionamento conjunto entre rotas, controladores, modelos e banco de dados. Esses testes contemplaram fluxos completos, incluindo autenticação, criação e alteração de reservas, regras de conflito e permissões de acesso.

Testes de Caixa-Preta (Exploratórios). Complementando os testes automatizados, foram realizados testes exploratórios na interface do sistema, avaliando o comportamento do software sem considerar sua estrutura interna. Esses testes permitiram identificar falhas de navegação, validações incorretas e inconsistências visuais, que foram ajustadas em ciclos curtos de correção.

Demonstrações para Coleta de Feedback. Como forma de validar informalmente os fluxos principais, foram realizadas demonstrações do sistema para os setores envolvidos no processo de agendamento. Embora essas interações não configurem testes de aceitação formais, permitiram identificar melhorias e ajustes que contribuíram para o refinamento da aplicação.

Quanto à documentação formal dos testes, optou-se por uma abordagem prática. As verificações realizadas tiveram como base direta os Requisitos Funcionais e os Casos de Uso definidos no projeto.

A combinação desses procedimentos permitiu assegurar o funcionamento interno do sistema e sua aderência aos requisitos definidos, mesmo diante da ausência de um ambiente institucional de homologação.

3.3 Planejamento e Arquitetura da Solução

A definição da arquitetura buscou alinhar as decisões técnicas com as diretrizes do setor de Tecnologia da Informação do HUMAP/UFMS, priorizando tecnologias já utilizadas ou suportadas pela instituição. No *backend*, adotou-se o *framework Laravel* [8], baseado em PHP: *Hypertext Preprocessor* [14], por sua aderência ao padrão MVC, robustez e mecanismos nativos de segurança. Para o *frontend*, optou-se pela biblioteca *React* [16], permitindo a construção de uma interface desacoplada (*client-side*) e com maior flexibilidade de evolução.

A aplicação foi estruturada em uma arquitetura desacoplada, na qual o *frontend* e o *backend* comunicam-se exclusivamente por meio de *Application Programming Interface (API) Representational State Transfer (REST)*. Para garantir consistência entre ambientes de desenvolvimento e futura implantação, utilizou-se *Docker* [3] e *Docker Compose* [4] como padrão de orquestração.

O controle de versão foi realizado com *Git* [22], utilizando repositório privado no *GitHub* [6], o que permitiu registro contínuo das evoluções, organização das etapas e rastreabilidade das modificações do projeto.

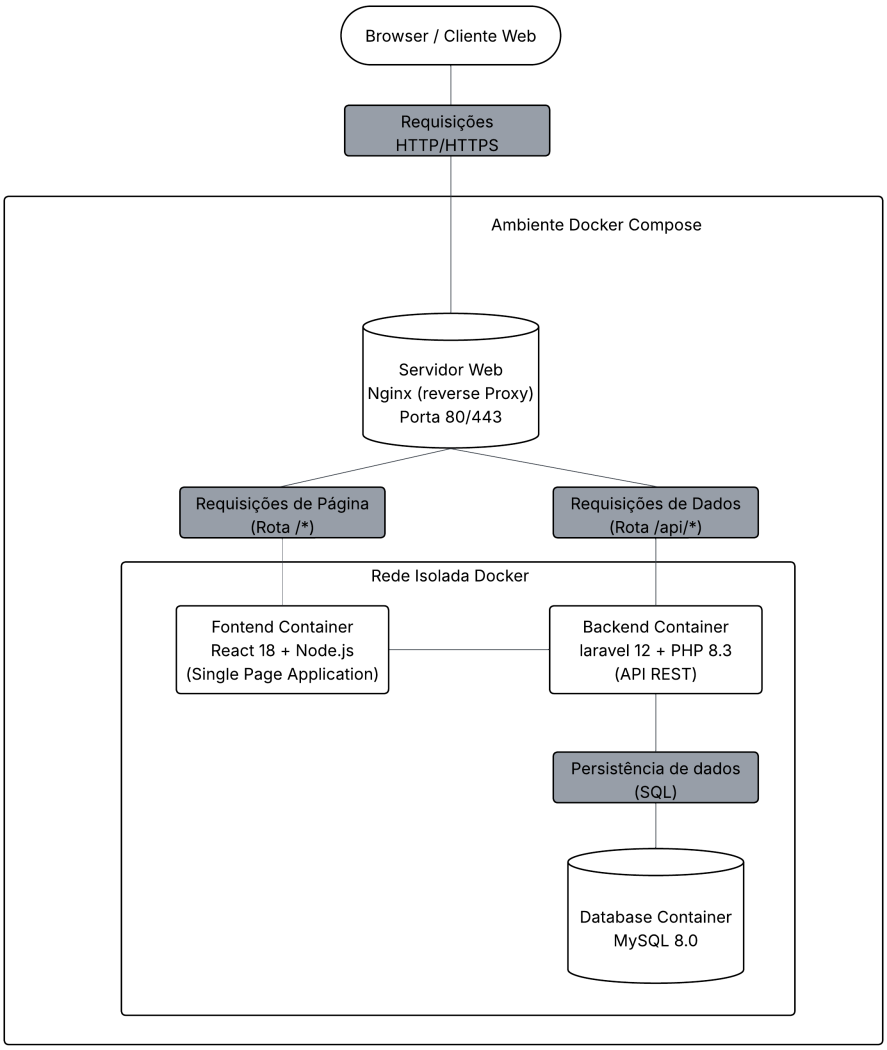
A Figura 3.1 representa a arquitetura geral do sistema HUBook, destacando a separação entre o frontend desenvolvido em *React*, o *backend* baseado na API REST construída com *Laravel* e o banco de dados.

3.4 Descrição Técnica da Arquitetura

O sistema HUBook foi desenvolvido seguindo uma arquitetura desacoplada cliente-servidor, onde o *frontend* (aplicação *React*) e o *backend* (API *Laravel*) são desenvolvidos e implantados de forma independente, comunicando-se através de uma API REST.

A arquitetura é composta pelos seguintes componentes principais:

Figura 3.1: Arquitetura Geral do Sistema HUBook



- ***Frontend*** - *Single Page Application* (SPA) desenvolvida em *React* 18 [16] com TypeScript [7], utilizando *Vite* [23] como *build tool*. A aplicação roda em *Node.js* [12] durante desenvolvimento e é servida através de *Nginx* [5] em produção.

- ***Backend*** - API REST desenvolvida em *Laravel 12* [8] (*PHP 8.3* [14]), seguindo padrão MVC. O *backend* processa requisições através de *PHP FastCGI Process Manager (PHP-FPM)* e comunica-se com o banco de dados *My Structured Query Language (MySQL)* [13].

- **Banco de Dados** - *MySQL 8.0* [13] armazena todos os dados persistentes do sistema, incluindo usuários, setores, salas, tipos de salas e reservas.

- **Servidor Web** - *Nginx* [5] atua como *reverse proxy*, direcionando requisições para o *frontend* ou *backend* conforme o caminho da URL.

- **Containerização** - Todo o ambiente é executado em *containers Docker* [3], orquestrados através de *Docker Compose* [4].

A comunicação entre *frontend* e *backend* é realizada através de requisições Hypertext Transfer Protocol (HTTP), com autenticação baseada em sessão utilizando Laravel Sanctum [9]. O *frontend* obtém um cookie Cross-Site Request Forgery (CSRF) antes de realizar requisições autenticadas, garantindo proteção contra ataques CSRF.

3.4.1 Principais Módulos do *Frontend*

O *frontend* React [16] é organizado em componentes, páginas, *hooks* e serviços, os quais são descritos a seguir:

- **Autenticação**: O `AuthContext` gerencia estado global de autenticação, fornecendo funções de *login*, *logout*, registro e atualização de perfil. Verifica automaticamente autenticação ao carregar a aplicação através da rota `/api/auth/me`.
- **Comunicação com API**: O serviço `apiService` centraliza todas as requisições HTTP para o *backend*, incluindo obtenção de cookie CSRF, tratamento de erros e formatação de requisições. *Hooks* customizados (`useSectors`, `useRooms`, `useReservations`, etc.) encapsulam lógica de busca e atualização de dados, utilizando TanStack Query [21] para *cache* e sincronização.
- **Componentes de Interface**: Componentes reutilizáveis são organizados por funcionalidade: componentes de formulário (reserva, perfil, *login*, registro); componentes de exibição (*cards* de reserva, lista de salas, calendário); componentes administrativos (aprovação de reservas, gerenciamento de salas); e componentes de UI base (botões, *inputs*, *dialogs*, *toasts*) utilizando *shadcn/ui* [19].
- **Roteamento**: React Router [18] gerencia navegação *client-side*, com rotas protegidas que verificam autenticação e permissões de administrador antes de renderizar componentes. Rotas públicas (*login*, registro, visualização de salas) são acessíveis sem autenticação.
- **Validação de Formulários**: React Hook Form [17] gerencia estado de formulários, enquanto *Zod* [10] valida *schemas TypeScript* [7]. Validação ocorre tanto no *frontend* (*feedback* imediato) quanto no *backend* (segurança), garantindo consistência e segurança dos dados.

3.4.2 Principais Módulos do *Backend*

O *backend Laravel* é organizado seguindo o padrão MVC, com os seguintes módulos principais.

- **Autenticação**: O módulo de autenticação utiliza *Laravel Sanctum* para gerenciamento de sessões. O *`AuthController`* implementa *login* (com suporte a e-mail ou CPF), registro, *logout*, recuperação de senha e atualização de perfil. A autenticação é *stateful*, utilizando sessões do *Laravel* para requisições do *frontend* SPA.

- **Gerenciamento de Usuários**: O `UserController` gerencia operações Create, Read, Update, Delete (CRUD) de usuários, disponível apenas para administradores. Implementa *soft delete*, permitindo restauração e exclusão permanente. Inclui filtros avançados por tipo, vínculo, busca textual e visualização de usuários deletados.

- **Gerenciamento de Espaços Físicos**: Três *controllers* gerenciam os espaços físicos: `SectorController` gerencia setores (CRUD para administradores, listagem pública); `RoomTypeController` gerencia tipos de salas (CRUD para administradores, listagem pública); e `RoomController` gerencia salas (CRUD para administradores, listagem e visualização públicas).

- **Sistema de Reservas**: O `ReservationController` é o núcleo do sistema, implementando criação, listagem, visualização, edição, cancelamento, aprovação e rejeição de reservas. Validações complexas são implementadas em `ReservationStoreRequest` e `ReservationUpdateRequest`, incluindo verificação de capacidade e conflitos de horário.

- **Autorização**: O sistema utiliza Policies do *Laravel* para controle de autorização: `ReservationPolicy` controla visualização, edição, cancelamento e aprovação/rejeição de reservas; `RoomPolicy` controla operações em salas (apenas administradores podem criar/editar/remover); `SectorPolicy` controla operações em setores (apenas administradores); e `UserPolicy` controla operações em usuários (apenas administradores).

- **Validação**: *Request Validators* customizados validam dados de entrada para cada operação, incluindo regras específicas de negócio. Validações complexas, como verificação de conflitos de horário e capacidade, são implementadas no método `withValidator()` dos *Form Requests*.

- **Resources**: API *Resources* formatam respostas da API, garantindo que apenas dados apropriados sejam retornados. `PublicReservationResource` oculta dados sensíveis para visualização pública, enquanto `ReservationResource` retorna informações completas para usuários autenticados.

3.4.3 Estrutura do Banco de Dados

O banco de dados do sistema é composto pelas seguintes entidades principais e seus relacionamentos.

- ***Users* (Usuários)**: Armazena informações dos usuários do sistema, incluindo nome, e-mail, CPF (único), telefone, vínculo (estudante, professor, funcionário-hu, outro), tipo (usuario, admin) e senha (*hash bcrypt*). Implementa *soft delete*, permitindo restauração de usuários deletados.

- ***Sectors* (Setores)**: Representa setores do hospital aos quais as salas pertencem. Possui nome e descrição opcional.

- ***Rooms* (Salas)**: Armazena informações das salas disponíveis para reserva, incluindo nome, setor (relacionamento com *Sectors*), tipo (sala, auditório, laboratório, outro),

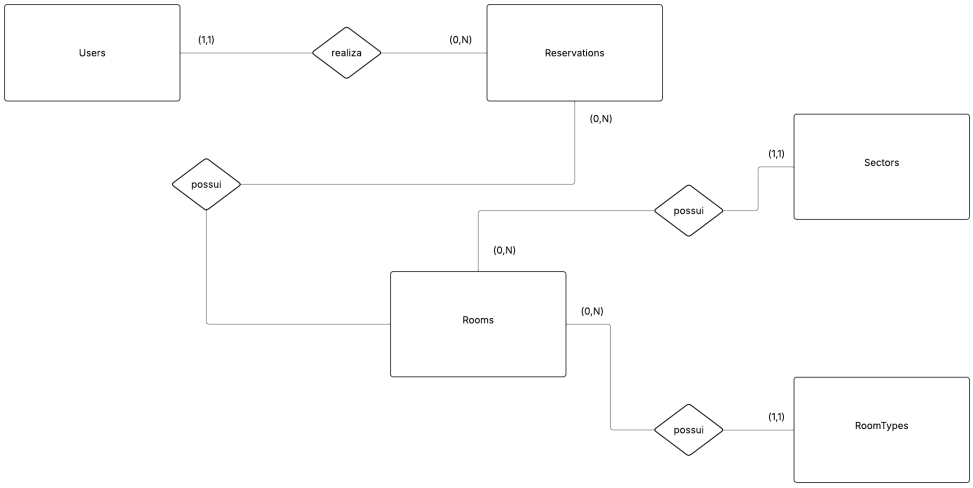
capacidade máxima, status (disponível, reservado, manutenção), descrição e recursos (*array JSON*).

- ***Reservations* (Reservas)**: Entidade central do sistema, armazena solicitações de reserva com informações sobre sala (relacionamento com *Rooms*), usuário solicitante (relacionamento com *Users*), título, data/hora de início e término, propósito, número de participantes, status (pendente, aprovado, recusado, cancelado), observações, motivo de rejeição (se aplicável) e dados de terceiros (quando reserva é para terceiro).
- ***RoomTypes* (Tipos de Salas)**: Tabela de referência para tipos de salas, permitindo gerenciamento centralizado de tipos disponíveis.

Os relacionamentos entre entidades são:

- ***Users*** → ***Reservations***: Um usuário pode ter múltiplas reservas (1:N).
- ***Sectors***→ ***Rooms***: Um setor pode ter múltiplas salas (1:N).
- ***Rooms***→ ***Reservations***: Uma sala pode ter múltiplas reservas (1:N).
- ***RoomTypes***→ ***Rooms***: Um tipo de espaço pode ser utilizado por múltiplas salas (1:N).

Figura 3.2: Diagrama Entidade-Relacionamento (DER) Conceitual



Fonte: Elaborada pelo autor (2025).

Conforme demonstra a Figura 3.2, o diagrama documenta as entidades, definindo os relacionamentos e as cardinalidades mínimas e máximas.

O banco de dados implementa índices para otimização de consultas frequentes: índice composto em `reservations(room_id, start_time, end_time)` para verificação eficiente de conflitos de horário; índice em `rooms(sector_id)` para consultas por setor; *constraints* de unicidade em `users(email)` e `users(cpf)`; e *foreign keys* com *cascade delete* para garantir integridade referencial.

3.4.4 Containerização e Infraestrutura

Todo o ambiente é containerizado utilizando *Docker Compose* [4], criando uma rede isolada onde todos os serviços comunicam-se através de nomes de serviço. Volumes persistentes garantem que dados do banco de dados não sejam perdidos ao reiniciar *containers*.

A configuração do *Nginx* [5] direciona requisições `/api/*` para o *backend Laravel* [8] e todas as outras requisições para o *frontend* React [16], permitindo que o *frontend* gerencie roteamento *client-side* enquanto o *backend* processa apenas requisições de API.

Essa arquitetura containerizada facilita desenvolvimento, testes, implantação (*deploy*) e manutenção, garantindo que o sistema seja portável, escalável e fácil de configurar em diferentes ambientes.

Capítulo 4

Resultados

Este capítulo apresenta os resultados obtidos com o desenvolvimento do sistema HUBook, destacando as funcionalidades implementadas, a consolidação da arquitetura planejada e o produto final entregue. Os resultados aqui expostos refletem a materialização do conjunto de requisitos levantados, das decisões técnicas adotadas e das etapas de refinamento realizadas durante o desenvolvimento.

4.1 Visão Geral do Sistema

O HUBook foi concluído como uma aplicação *web*, responsiva e funcional, capaz de centralizar o processo de agendamento de espaços físicos do HUMAP/UFMS. O sistema abrange toda a cadeia de uso prevista: visitantes podem visualizar informações gerais, usuários autenticados realizam solicitações e acompanham suas reservas, e administradores gerenciam setores, salas, usuários e o fluxo completo de aprovação.

A porta de entrada para a utilização do sistema é apresentada na Figura 4.1. A página inicial foi projetada para oferecer acesso rápido à navegação, estatísticas de uso e *cards* informativos com as principais salas disponíveis, facilitando a localização visual dos recursos.

A arquitetura desacoplada planejada foi implementada integralmente: o *frontend*, desenvolvido em *React* [16], consome exclusivamente a *Application Programming Interface* (API) construída com *Laravel* [8]. Essa separação permitiu uma interface dinâmica e fluida, enquanto o *backend* concentra regras de negócio, segurança e persistência de dados. O resultado é um sistema escalável, organizado e preparado para futuras integrações institucionais.

Para compreender a dinâmica de interação entre os diferentes perfis de usuário e o sistema, a Figura 4.2 detalha o fluxo completo de criação de reserva, mapeando desde a solicitação inicial feita pelo usuário até a etapa final de aprovação ou rejeição pelo administrador.

Ao longo do desenvolvimento, diversas demonstrações foram realizadas aos setores responsáveis pelo fluxo de reservas, o que permitiu coletar *feedback* contínuo e ajustar a

experiência de uso, navegação e organização visual da aplicação. Esses ciclos contribuíram para tornar o produto final mais aderente ao contexto hospitalar, que exige clareza, rapidez e padronização.

4.2 Módulo de Autenticação e Gestão de Usuários

A adaptação necessária no escopo, devido à impossibilidade de integrar com o sistema de autenticação institucional, resultou no desenvolvimento de um módulo próprio e completo de gestão de identidades. Esse módulo incorpora mecanismos essenciais para um ambiente seguro e controlado.

A página de *Login* (Figura 4.3) permite autenticação por *e-mail* ou CPF, com validações automáticas e *feedback* imediato ao usuário. O fluxo de registro (Figura 4.4) implementa validações de integridade, como verificação matemática do CPF, formatação automática e confirmação de senha.

Além do registro e acesso, o sistema garante autonomia ao usuário através da página de Perfil (Figura 4.5). Nesta interface, é possível realizar a atualização de informações pessoais e a alteração de senha, mantendo os dados cadastrais sempre correntes.

Para administradores, foi disponibilizado um painel completo de gestão de usuários, incluindo funcionalidades de criar novos usuários, edição e exclusão lógica (*soft delete*), garantindo que o histórico do usuário seja preservado mesmo após sua remoção operacional. Esse recurso é essencial em ambientes institucionais, onde registros de acesso precisam ser mantidos para auditoria.

4.3 Gerenciamento de Reservas e Interface de Calendário

O módulo de reservas representa o núcleo do HUBook e concentra as regras de negócio mais importantes do projeto. A lógica de prevenção de conflitos, implementada no *backend*, foi cuidadosamente construída e testada para impedir sobreposições de horários — problema central identificado no levantamento de requisitos.

A interface de calendário (Figura 4.6) permite visualizar reservas por setor, tipo de sala e período, facilitando a interpretação da ocupação dos espaços. A visualização em formato agenda, semanal e mensal contribui para atender diferentes perfis de uso. Todo o fluxo foi pensado para oferecer clareza, especialmente considerando o ambiente hospitalar, onde múltiplas equipes compartilham espaços.

O formulário de criação de reservas (Figura 4.7) foi projetado para ser objetivo e intuitivo, permitindo que o usuário selecione facilmente o setor, o espaço desejado e o intervalo de tempo. Além disso, o sistema permite que o usuário crie solicitações em nome de terceiros, atendendo à necessidade identificada entre setores que realizam reservas para professores, colaboradores ou visitantes.

Após a criação do pedido, o usuário dispõe de uma área dedicada para o acompanhamento pessoal, intitulada "Minhas Solicitações"(Figura 4.8). Esta tela organiza os pedidos em abas por *status* e oferece funcionalidade de busca, permitindo um controle rápido sobre o andamento das reservas.

O Diagrama Entidade–Relacionamento Físico (Figura 4.9) confirma que a estrutura do banco foi implementada conforme planejado, suportando adequadamente relações complexas entre setores, salas, tipos de espaço, usuários e reservas. Essa modelagem foi essencial para garantir consistência e evitar registros duplicados ou inconsistentes.

4.4 Painel Administrativo

O painel destinado ao administrador consolida todas as funcionalidades necessárias para a gestão integral do sistema. A Figura 4.10 apresenta a interface de gerenciamento de reservas, onde é possível visualizar solicitações pendentes, aprovar ou rejeitar pedidos, registrar justificativas, cancelar reservas e consultar o histórico completo.

Além do controle sobre os agendamentos, o administrador possui ferramentas para a gestão da infraestrutura física cadastrada. A Figura 4.11 demonstra o painel administrativo para gerenciamento de salas, que permite a criação, edição e remoção de espaços físicos conforme a necessidade da instituição.

Esse módulo foi projetado para reduzir o esforço manual do setor responsável pelo agendamento, permitindo que cada ação administrativa seja executada de forma rápida e transparente. Como resultado, o fluxo antigo, descentralizado e suscetível a falhas, foi substituído por uma interface unificada e padronizada.

4.5 Entrega Técnica e Prontidão do Sistema

O sistema completo foi concluído em uma estrutura totalmente containerizada, utilizando *Docker* [3] e *Docker Compose* [4], permitindo que o ambiente seja reproduzido fielmente pela equipe de Tecnologia da Informação (TI) do HUMAP. Essa abordagem elimina inconsistências entre o ambiente de desenvolvimento e o ambiente de produção, facilitando significativamente a futura implantação institucional.

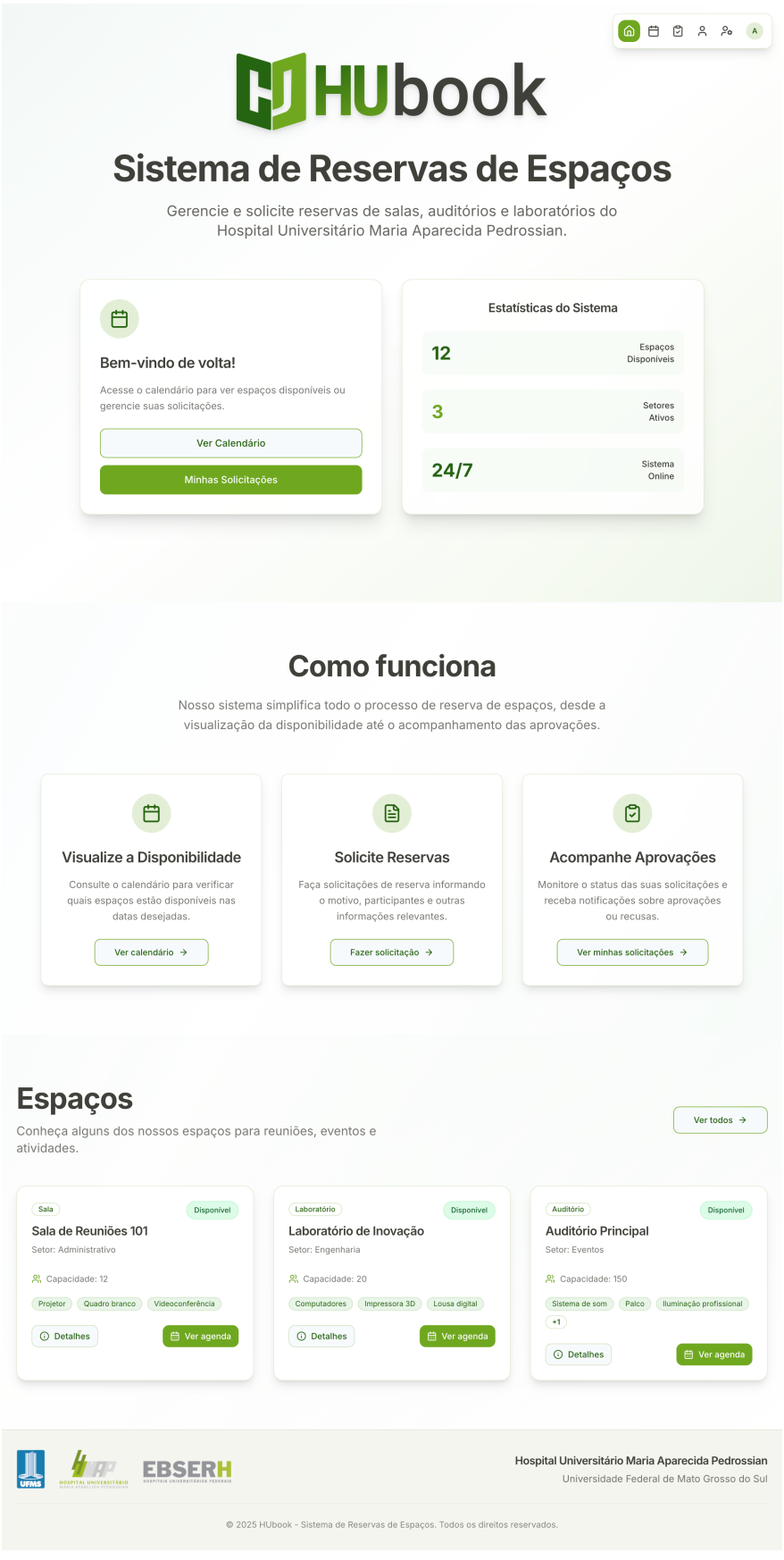
Quanto aos testes realizados, os resultados confirmam a estabilidade da solução:

- **Testes unitários** asseguraram o comportamento correto de partes críticas do código.
- **Testes de integração** validaram regras de negócio como prevenção de conflitos, controle de permissões e fluxo de autenticação.
- **Testes exploratórios** permitiram identificar falhas de navegação e inconsistências visuais, posteriormente corrigidas.

Embora não tenham sido conduzidos testes formais de aceitação devido à indisponibilidade de ambiente institucional de homologação, as demonstrações realizadas aos setores envolvidos contribuíram para aprimorar fluxos e detalhes de funcionalidade durante o desenvolvimento.

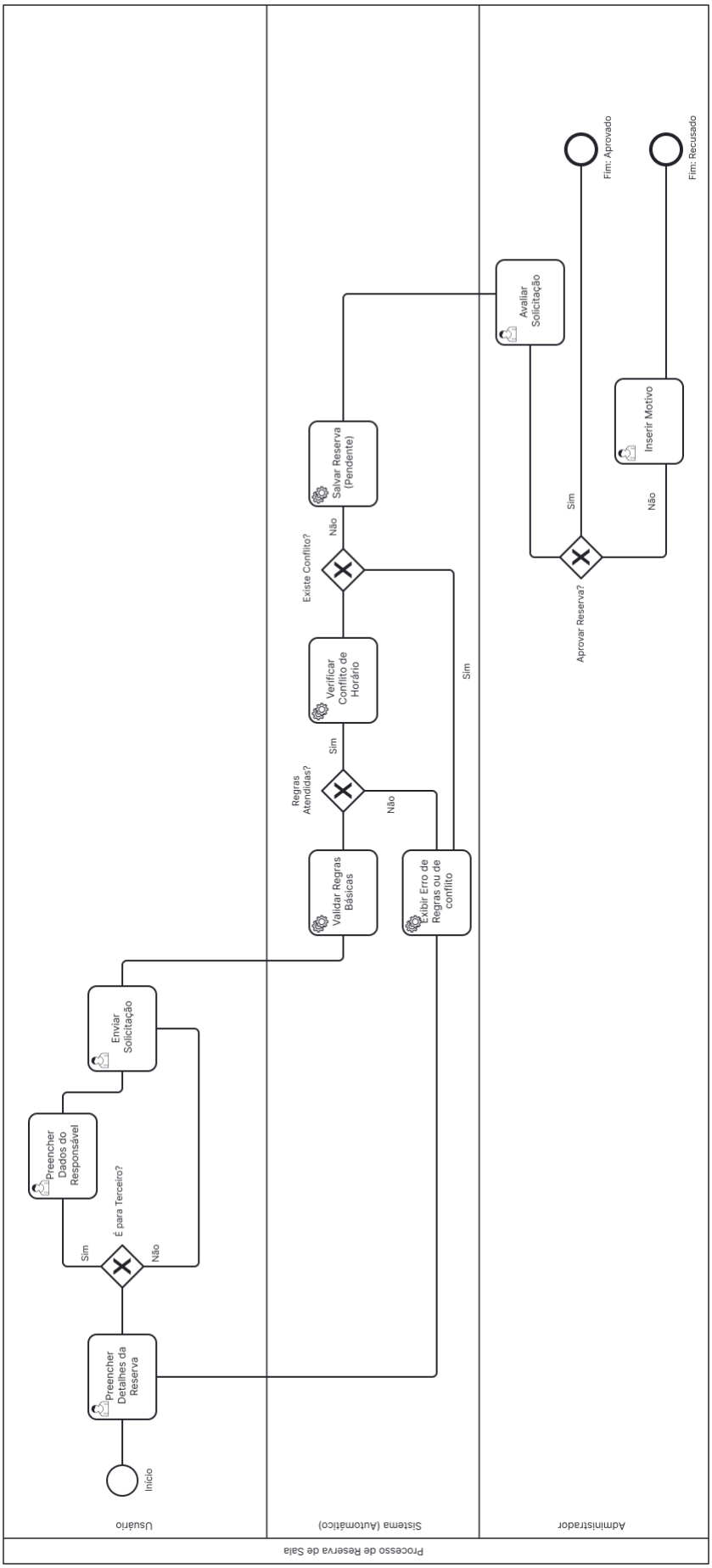
Assim, o artefato final não consiste apenas no código-fonte, mas em um sistema completo, funcional e preparado para futura implantação no HUMAP/UFMS, atendendo aos requisitos mapeados e oferecendo uma base sólida para evoluções futuras.

Figura 4.1: Página Inicial do Sistema HUBook



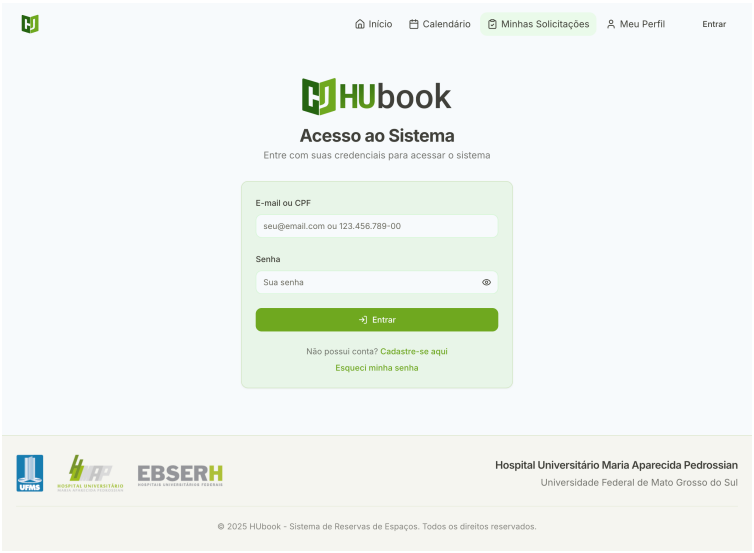
Fonte: Elaborada pelo autor (2025).

Figura 4.2: Fluxo de Criação e Aprovação de Reserva



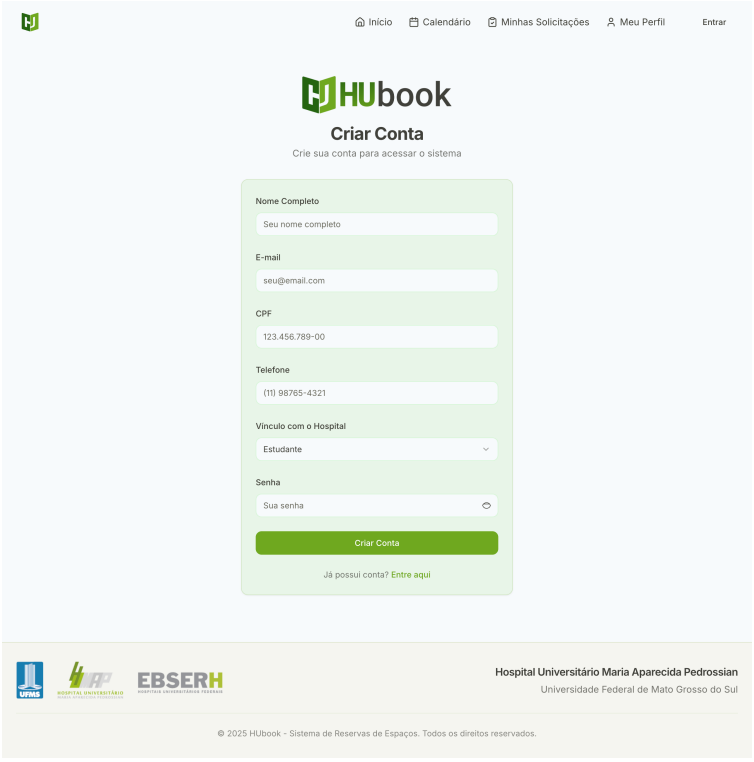
Fonte: Elaborada pelo autor (2025).

Figura 4.3: Página de *Login*



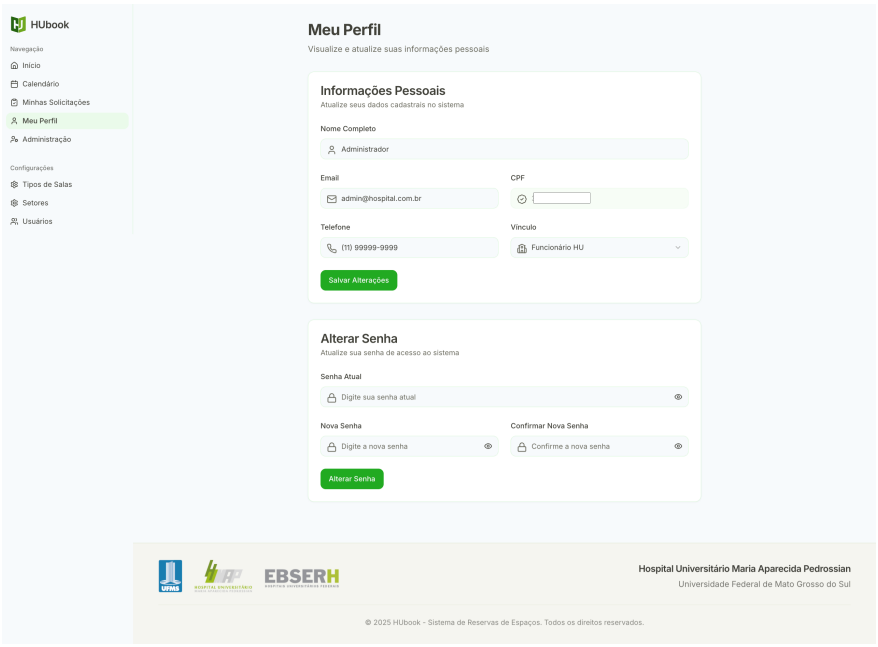
Fonte: Elaborada pelo autor (2025).

Figura 4.4: Página de Registro de Usuário



Fonte: Elaborada pelo autor (2025).

Figura 4.5: Página de Perfil do Usuário

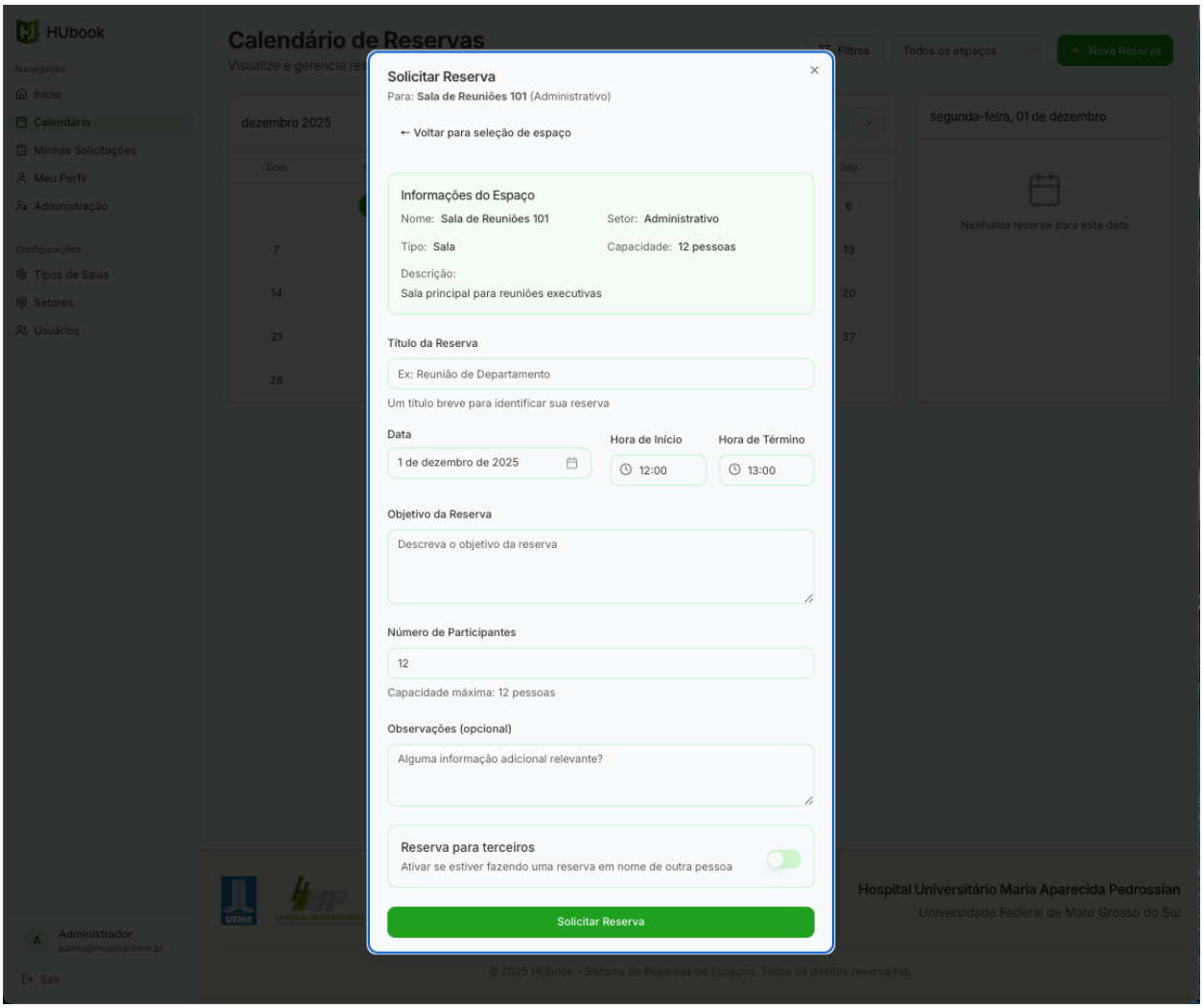


Fonte: Elaborada pelo autor (2025).

Figura 4.6: Página do Calendário de Reservas

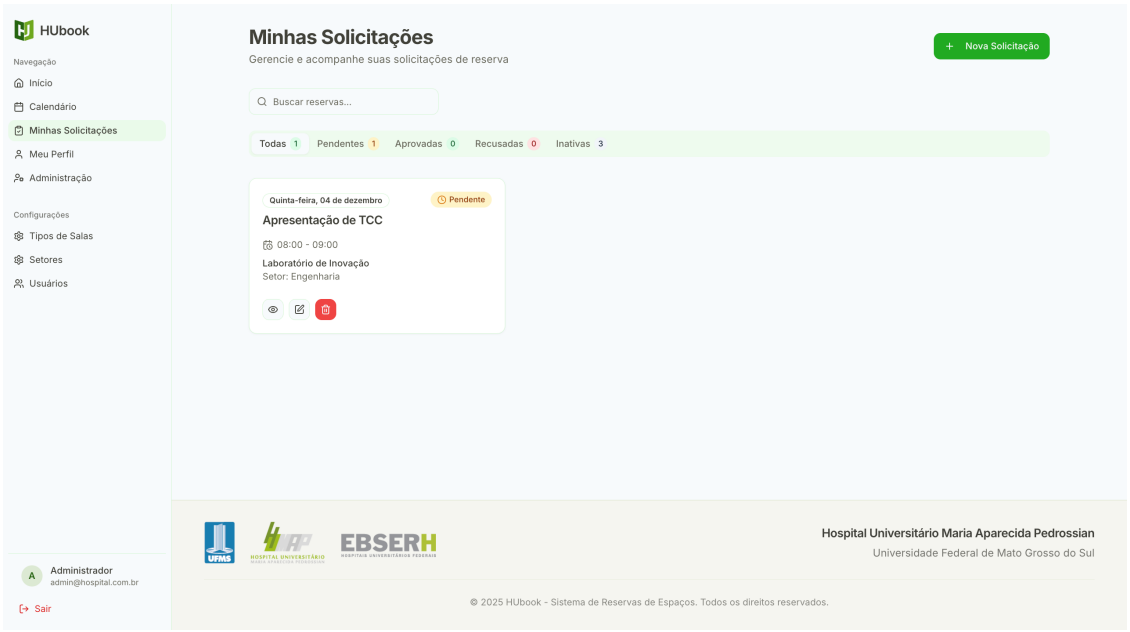
Fonte: Elaborada pelo autor (2025).

Figura 4.7: Página do Formulário de Criação de Reserva



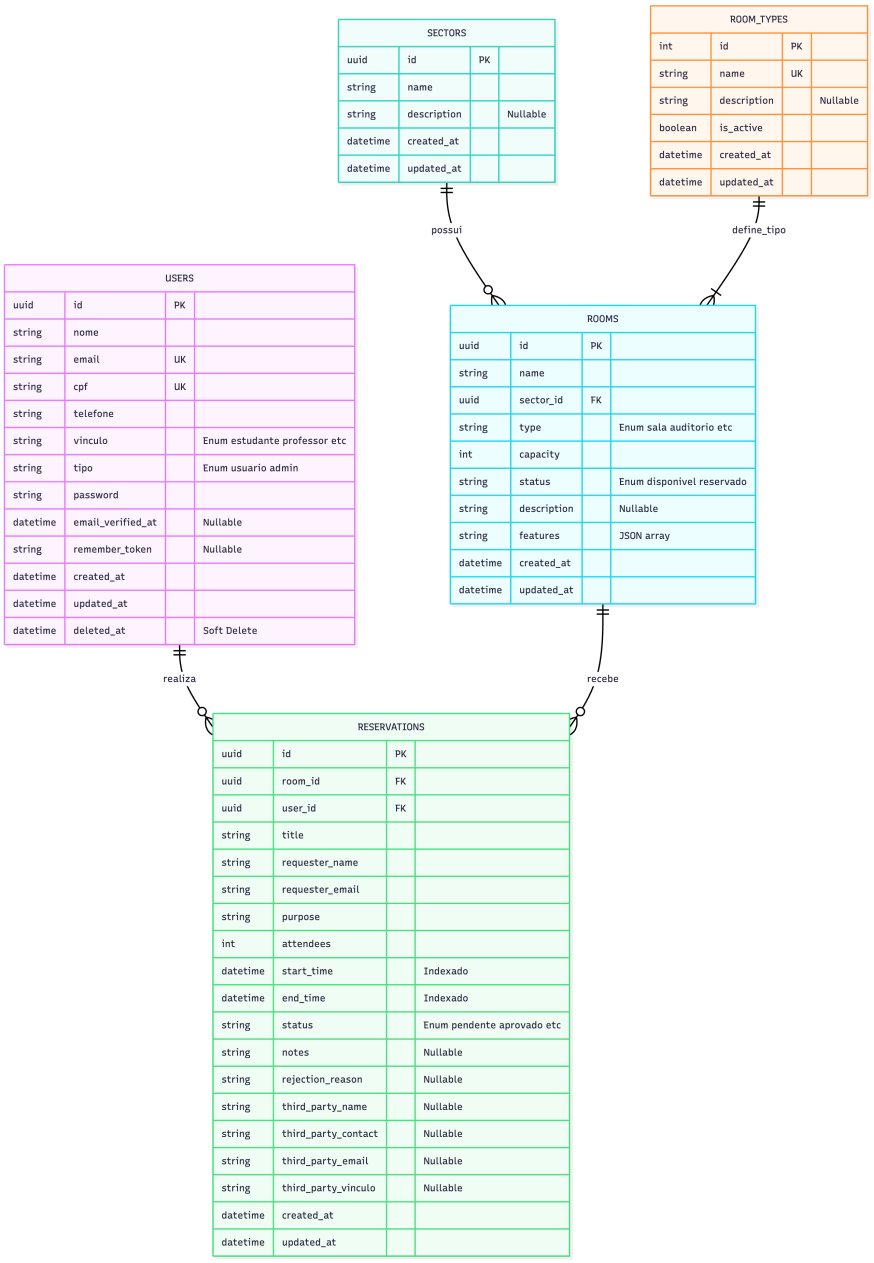
Fonte: Elaborada pelo autor (2025).

Figura 4.8: Página Minhas Solicitações



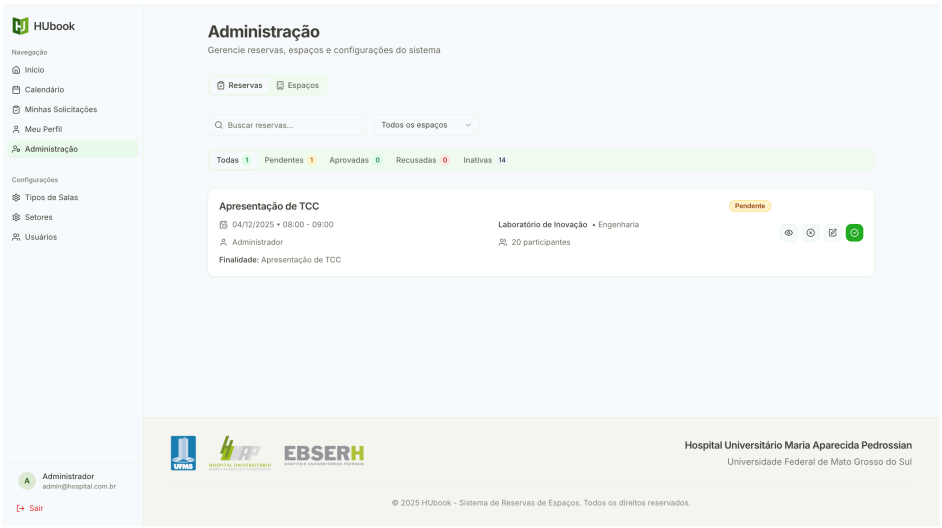
Fonte: Elaborada pelo autor (2025).

Figura 4.9: Diagrama de Entidades-Relacionamentos Físico



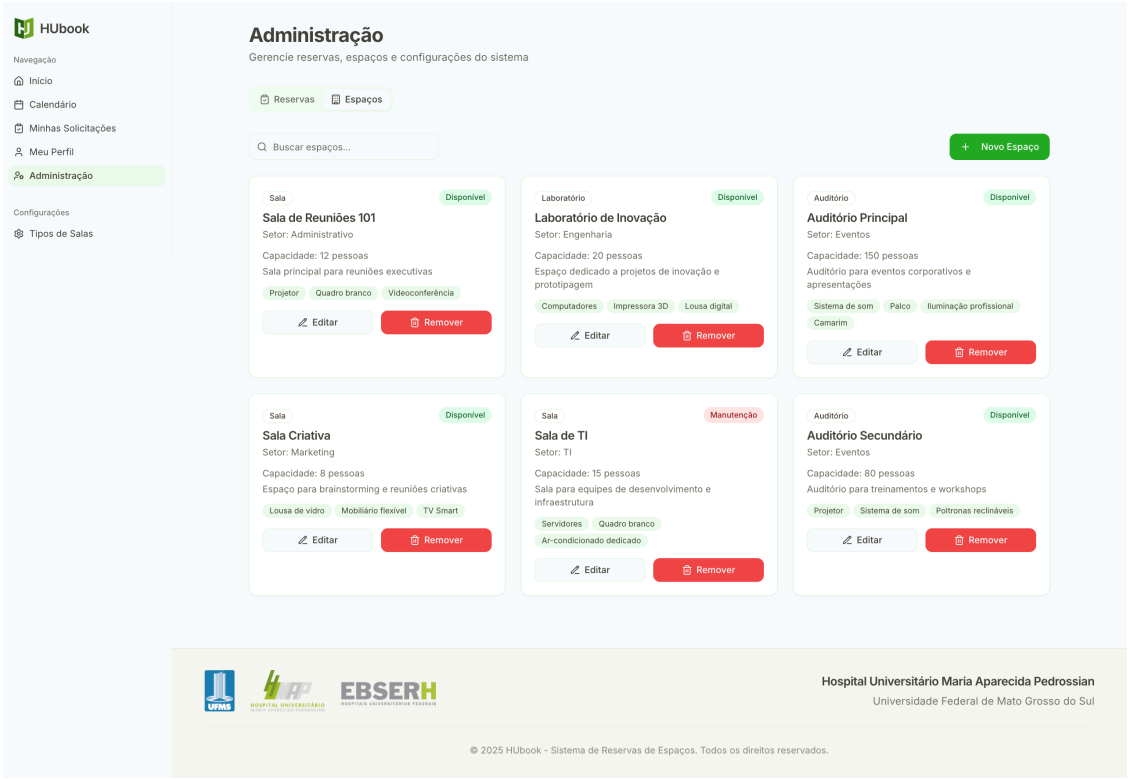
Fonte: Elaborada pelo autor (2025).

Figura 4.10: Página do Painel Administrativo - Gerenciamento de Reservas



Fonte: Elaborada pelo autor (2025).

Figura 4.11: Página do Painel Administrativo - Gerenciamento de Salas



Fonte: Elaborada pelo autor (2025).

Capítulo 5

Considerações Finais

Este capítulo apresenta uma síntese dos objetivos alcançados, das contribuições do sistema HUBook, das limitações encontradas no desenvolvimento e das perspectivas de evolução futura da solução.

5.1 Contribuições

O desenvolvimento do HUBook permitiu alcançar o objetivo geral proposto: construir um sistema *web* capaz de centralizar e padronizar o processo de agendamento de espaços físicos no HUMAP/UFMS. O sistema resultante oferece uma aplicação funcional, alinhada às necessidades mapeadas.

Os objetivos específicos foram amplamente atendidos, incluindo a implementação do módulo de autenticação, gerenciamento de usuários e espaços, criação de reservas com validações automáticas, interface de calendário, painel administrativo, área pessoal para usuários, além de uma arquitetura desacoplada e containerizada. Os requisitos funcionais e não funcionais levantados foram implementados conforme escopo definido e encontram-se descritos no Apêndice A.

O HUBook contribui diretamente para a modernização dos processos administrativos do HUMAP/UFMS. A centralização das solicitações reduz a fragmentação anteriormente existente e a automação das verificações de conflito de horário elimina falhas recorrentes do modelo anterior. A interface responsiva e organizada favorece a experiência de uso, enquanto as tecnologias adotadas fornecem uma base sólida para evoluções futuras, incluindo integrações institucionais e funcionalidades adicionais como relatórios e notificações.

5.2 Limitações e Trabalhos Futuros

Algumas limitações foram identificadas ao longo do projeto. Não foi possível integrar o sistema com o mecanismo institucional de autenticação, resultando na necessidade

de desenvolver um módulo próprio. A ausência de um ambiente oficial de homologação impossibilitou a realização de testes formais com usuários finais, restringindo a validação a testes automatizados e testes exploratórios conduzidos em ambiente local. Além disso, algumas funcionalidades inicialmente planejadas, como notificações automáticas, relatórios gerenciais, integração com calendários externos e documentação destinada aos usuários — permaneceram fora do escopo desta versão.

Com base nessas limitações, diversos aprimoramentos podem ser realizados em trabalhos futuros, como:

- Integração com o sistema institucional de autenticação do HUMAP/UFMS;
- implementação de notificações por e-mail; desenvolvimento de relatórios gerenciais e *dashboards*;
- integração com serviços externos de calendário; criação de sistema de avaliação de salas; auditoria detalhada de ações; suporte a reservas recorrentes; bloqueio administrativo de períodos; e evolução para um aplicativo *mobile* ou *Progressive Web App (PWA)*.
- Também se destacam oportunidades de melhorias contínuas de acessibilidade e usabilidade.

5.3 Considerações Finais

O desenvolvimento do HUBook possibilitou a aplicação prática de conceitos de engenharia de software, desenvolvimento *web* e metodologias ágeis na solução de um problema institucional concreto. O sistema demonstra a viabilidade de empregar tecnologias contemporâneas, arquitetura desacoplada e práticas de desenvolvimento incremental para construir aplicações alinhadas às demandas de ambientes acadêmicos e hospitalares.

O artefato entregue atende ao conjunto de requisitos definidos e apresenta um sistema funcional, organizado e preparado para implantação futura. A arquitetura implementada favorece manutenção, escalabilidade e expansão, garantindo a longevidade da solução. A apresentação do projeto na 3ª Jornada Científica do HUMAP-UFMS, onde recebeu destaque como 2º melhor trabalho de iniciação tecnológica, reforça seu potencial de aplicação e o interesse institucional na continuidade da proposta.

Este trabalho estabelece uma base para futura implantação institucional, continuidade em pesquisas futuras e possível registro de propriedade intelectual, contribuindo para a modernização dos processos internos do HUMAP/UFMS e para a formação acadêmica e profissional envolvida em seu desenvolvimento.

Capítulo 6

Referências Bibliográficas

[1] BECK, Kent; ANDRES, Cynthia. *Programação Extrema (XP) Explicada: Acolha as mudanças. 2. ed.* [s.n.], Porto Alegre, RS, Brasil, 2004.

[2] BERGMANN, Sebastian. PHPUnit: The PHP Testing Framework. Disponível em: <https://phpunit.de/index.html>. Acesso em: 5 dez. 2025.

[3] DOCKER, INC. Docker: Accelerated Container Application Development. Disponível em: <https://www.docker.com/>. Acesso em: 5 dez. 2025.

[4] DOCKER, INC. Docker Compose. Disponível em: <https://docs.docker.com/compose/>. Acesso em: 5 dez. 2025.

[5] F5 NGINX Products. Disponível em: <https://www.f5.com/products/nginx>. Acesso em: 5 dez. 2025.

[6] GitHub · A mudança é constante. O GitHub mantém você à frente. Disponível em: <https://github.com/?locale=pt-BR>. Acesso em: 5 dez. 2025.

[7] JavaScript With Syntax For Types. Disponível em: <https://www.typescriptlang.org/>. Acesso em: 5 dez. 2025.

[8] Laravel - The PHP Framework For Web Artisans. Disponível em: <https://laravel.com>. Acesso em: 5 dez. 2025.

[9] Laravel Sanctum - Laravel 12.x - The PHP Framework For Web Artisans. Disponível em: <https://laravel.com/docs/12.x/sanctum>. Acesso em: 5 dez. 2025.

[10] MCDONNELL, Colin. Zod - typescript-first schema validation. Disponível em: <https://zod.dev/>. Acesso em: 5 dez. 2025.

[11] MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. *The Art of Software Testing. 3. ed.* Wiley, [s.l.], 2011. Obra sem tradução oficial para o português.

[12] Node.js — Run JavaScript Everywhere. Disponível em: <https://nodejs.org/pt>. Acesso em: 5 dez. 2025.

[13] ORACLE CORPORATION. The world's most popular open source database. Disponível em: <https://www.mysql.com/>. Acesso em: 5 dez. 2025.

[14] PHP. Disponível em: <https://www.php.net/index.php>. Acesso em: 5 dez. 2025.

[15] PRESSMAN, Roger S.; MAXIM, Bruce R. *Engenharia de Software: uma abordagem profissional. 9. ed.* Amgh, Porto Alegre, RS, 2021.

[16] React - a javascript library for building user interfaces. Disponível em: <https://react.dev/>. Acesso em: 5 dez. 2025.

[17] React Hook Form - performant, flexible and extensible form library. Disponível em: <https://react-hook-form.com/>. Acesso em: 5 dez. 2025.

[18] React Router Official Documentation. Disponível em: <https://reactrouter.com/>. Acesso em: 5 dez. 2025.

[19] SHADCN. The Foundation for your Design System - shadcn/ui. Disponível em: <https://ui.shadcn.com/>. Acesso em: 5 dez. 2025.

[20] SOMMERVILLE, Ian. *Engenharia de Software. 10. ed.* [s.n.], Brasil, 2019.

[21] TanStack Query. Disponível em: <https://tanstack.com/query/latest>. Acesso em: 5 dez. 2025.

[22] TORVALDS, Linus. Git - distributed version control system. Disponível em: <https://git-scm.com/>. Acesso em: 5 dez. 2025.

[23] YOU, Evan. Vite - next generation frontend tooling. Disponível em: <https://vite.dev>. Acesso em: 5 dez. 2025.

Apêndice A

Documentação do Sistema

Este apêndice apresenta a documentação do sistema HUBook, incluindo requisitos funcionais, requisitos não funcionais e tabela resumo dos casos de uso.

Requisitos Funcionais

Os requisitos funcionais descrevem o que o sistema deve fazer para atender as necessidades dos usuários. A seguir, são apresentados os requisitos funcionais implementados no sistema HUBook, organizados por categoria.

Autenticação e Gerenciamento de Usuários. O sistema implementa um conjunto completo de funcionalidades relacionadas à autenticação e gestão de usuários:

RF-01 O sistema deve permitir *login* utilizando e-mail ou CPF como identificador, juntamente com senha.

RF-02 O sistema deve permitir cadastro de novos usuários, coletando nome, e-mail, CPF, telefone, vínculo com o HUMAP/UFMS e senha.

RF-03 O sistema deve validar CPF durante o cadastro.

RF-04 O sistema deve aplicar máscaras automáticas para CPF e telefone durante o preenchimento de formulários.

RF-05 O sistema deve permitir recuperação de senha através de link enviado por e-mail.

RF-06 O sistema deve permitir atualização de perfil pessoal (nome, e-mail, telefone, vínculo).

RF-07 O sistema deve permitir alteração de senha, exigindo validação da senha atual.

RF-08 O sistema deve permitir que administradores gerenciem usuários (criar, editar, remover, restaurar).

RF-09 O sistema deve implementar *soft delete* para usuários, permitindo restauração posterior.

RF-10 O sistema deve impedir que administradores deletem permanentemente a si mesmos.

Gerenciamento de Espaços Físicos. O sistema oferece funcionalidades completas para administração de espaços físicos:

RF-11 O sistema deve permitir cadastro e edição de setores, com nome e descrição.

RF-12 O sistema deve permitir cadastro e edição de tipos de salas (sala, auditório, laboratório, outro).

RF-13 O sistema deve permitir cadastro e edição de salas, incluindo nome, setor, tipo, capacidade, status, descrição e recursos.

RF-14 O sistema deve impedir exclusão de setores que possuem salas associadas.

RF-15 O sistema deve impedir exclusão de tipos de salas que possuem salas utilizando-os.

RF-16 O sistema deve permitir visualização pública de salas disponíveis, com filtros por setor, tipo e status.

Sistema de Reservas. O núcleo do sistema consiste no módulo de reservas, que implementa validações automáticas e gestão completa:

RF-17 O sistema deve permitir criação de solicitações de reserva por usuários autenticados.

RF-18 O sistema deve validar automaticamente que a data/hora de início é futura.

RF-19 O sistema deve validar automaticamente que a data/hora de término é posterior ao início.

RF-20 O sistema deve validar automaticamente que o número de participantes não excede a capacidade da sala.

RF-21 O sistema deve verificar automaticamente conflitos de horário com outras reservas aprovadas ou pendentes.

RF-22 O sistema deve exibir mensagens detalhadas quando há conflitos de horário, incluindo horários conflitantes formatados.

RF-23 O sistema deve permitir criação de reservas em nome de terceiros, coletando informações do responsável.

RF-24 O sistema deve tornar todos os campos de terceiro obrigatórios quando a opção de reserva para terceiro é selecionada.

RF-25 O sistema deve criar reservas com status inicial "pendente".

RF-26 O sistema deve permitir que o usuário proprietário edite suas reservas (pendentes ou aprovadas), desde que a data e hora de início ainda não tenham sido ultrapassadas.

RF-27 O sistema deve permitir edição de qualquer reserva futura por administradores.

RF-28 O sistema deve bloquear a edição de reservas que já foram iniciadas, que estejam marcadas como inativas ou que possuam status "cancelado".

RF-29 O sistema deve alterar automaticamente o status para "pendente"após qualquer modificação na reserva, exigindo nova aprovação.

RF-30 O sistema deve permitir cancelamento de reservas pendentes ou aprovadas que ainda não passaram.

RF-31 O sistema deve realizar cancelamento através de *soft delete*, alterando status para "cancelado".

RF-32 O sistema deve permitir que administradores aprovem reservas pendentes.

RF-33 O sistema deve permitir que administradores rejeitem reservas pendentes, exigindo motivo obrigatório.

RF-34 O sistema deve salvar motivo de rejeição junto com a reserva recusada.

RF-35 O sistema deve permitir visualização de calendário público com reservas aprovadas.

RF-36 O sistema deve ocultar dados sensíveis (e-mail, telefone, dados de terceiros) na visualização pública.

RF-37 O sistema deve permitir visualização de calendário privado para usuários autenticados, com todas as reservas.

RF-38 O sistema deve oferecer filtros no calendário por setor, tipo de sala e sala específica.

RF-39 O sistema deve permitir acompanhamento pessoal de reservas na página "Minhas Solicitações".

RF-40 O sistema deve separar reservas em ativas (futuras) e inativas (passadas) na página de solicitações.

RF-41 O sistema deve permitir filtragem de reservas por status (pendentes, aprovadas, recusadas, canceladas, inativas).

RF-42 O sistema deve permitir busca de reservas por título, nome da sala ou propósito.

RF-43 O sistema deve ordenar reservas por data (ativas: mais próximas primeiro, inativas: mais recentes primeiro).

Requisitos Não Funcionais

Os requisitos não funcionais especificam como o sistema deve se comportar em relação a aspectos qualitativos, organizados nas seguintes categorias:

Usabilidade. O sistema prioriza experiência do usuário através de interface intuitiva e responsiva:

RNF-01 O sistema deve apresentar interface intuitiva e de fácil navegação.

RNF-02 O sistema deve ser responsivo, adaptando-se adequadamente a diferentes tamanhos de tela (*desktop*, *tablet*, *smartphone*).

RNF-03 O sistema deve fornecer *feedback* claro para todas as ações do usuário (sucesso, erro, carregamento).

RNF-04 O sistema deve exibir mensagens de erro descritivas e orientativas, ajudando o usuário a corrigir problemas.

Desempenho. O sistema foi projetado para oferecer respostas rápidas e suportar múltiplos usuários:

RNF-05 O sistema deve responder a operações comuns (*login*, listagem, criação de reserva) em tempo adequado (menos de 2 segundos).

RNF-06 O sistema deve suportar múltiplos usuários simultâneos sem degradação significativa de desempenho.

RNF-07 O sistema deve implementar paginação em listagens que podem retornar muitos resultados.

Segurança. A segurança é garantida através de múltiplas camadas de proteção:

RNF-08 O sistema deve utilizar autenticação baseada em *tokens* seguros para validar a identidade do usuário.

RNF-09 O sistema deve armazenar senhas com criptografia aplicada, nunca em texto plano.

RNF-10 O sistema deve implementar autorização baseada em *roles* (usuário comum e administrador).

RNF-11 O sistema deve validar e sanitizar dados de entrada tanto no *frontend* quanto no *backend* para prevenir XSS.

RNF-12 O sistema deve implementar medidas de segurança contra injeção de código malicioso e falsificação de solicitações.

RNF-13 O sistema deve ocultar informações sensíveis em respostas públicas da API.

RNF-14 O sistema deve sempre retornar mensagem genérica em recuperação de senha, não expondo se e-mail existe.

Manutenibilidade. O código segue padrões estabelecidos para facilitar manutenção futura:

RNF-15 O sistema deve seguir padrões estabelecidos pelos *frameworks frontend* e *backend*.

RNF-16 O sistema deve separar claramente responsabilidades entre camadas.

RNF-17 O sistema deve documentar código complexo e regras de negócio importantes.

Disponibilidade e Implantação. O sistema foi projetado para facilitar implantação e garantir disponibilidade:

RNF-18 O sistema deve ser acessível via navegador *web*, sem necessidade de instalação de software adicional.

RNF-19 O sistema deve ser containerizado utilizando *Docker*, garantindo consistência de execução entre ambientes *Linux*, *Windows* e *macOS*.

RNF-20 O sistema deve garantir persistência de dados através de volumes *Docker* persistentes.

Quadro Resumo dos Casos de Uso

O Quadro A.1 apresenta um resumo dos casos de uso implementados no sistema HUBook.

Quadro A.1: Resumo dos Casos de Uso do Sistema HUBook

ID	**Nome**	**Ator**	**Objetivo**
CU-01	Realizar *Login*	Usuário, Admin, Visitante	Autenticar-se no sistema utilizando e-mail ou CPF e senha
CU-02	Registrar Nova Conta	Visitante	Criar nova conta de usuário no sistema
CU-03	Solicitar Reserva de Sala	Usuário, Admin	Criar solicitação de reserva de espaço físico
CU-04	Visualizar Minhas Reservas	Usuário, Admin	Acompanhar reservas pessoais com filtros e busca
CU-05	Editar Reserva	Usuário (Próprias Reservas), Admin (Qualquer Reserva)	Modificar dados de reserva futuras, revertendo status para pendente
CU-06	Cancelar Reserva	Usuário (Próprias Reservas), Admin (Qualquer Reserva)	Cancelar reserva futura, liberando horário da sala
CU-07	Recuperar Senha	Usuário, Visitante	Redefinir senha através de link enviado por e-mail
CU-08	Atualizar Perfil	Usuário, Admin	Modificar informações pessoais e alterar senha
CU-09	Aprovar Reserva	Admin	Aprovar solicitação de reserva pendente
CU-10	Rejeitar Reserva	Admin	Rejeitar solicitação de reserva com motivo obrigatório
CU-11	Gerenciar Salas	Admin	Criar, editar e remover salas do sistema
CU-12	Gerenciar Setores	Admin	Criar, editar e remover setores do sistema
CU-13	Gerenciar Usuários	Admin	Criar, editar, remover e restaurar usuários
CU-14	Visualizar solicitações de Reservas	Admin	Visualizar todas as solicitações de reservas.
CU-15	Visualizar Salas Disponíveis	Visitante, Usuário, Admin	Consultar informações sobre salas disponíveis
CU-16	Visualizar Reservas Públicas	Visitante, Usuário, Admin	Visualizar reservas públicas com dados limitados