Sistema para Gestão de Bens Patrimoniais da Faculdade de Computação-UFMS

FACOM-UFMS
Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Autora: Amanda Vieira Costa

Orientadora: Profa. Dra. Liana Dessandre Duenha Garanhani

Trabalho de Conclusão de Curso.

09 de dezembro de 2025

Sumário

Lista de Siglas e Abreviaturas

API *Application Programming Interface*. 14, 19, 21–23, 33, 34

CDD *Component-Driven Development*. 14

CoC *Convention over configuration*. 13

CSS *Cascading Style Sheets*. 14

CSV *Comma-Separated Values*. 17, 18, 21, 33

CVCS *Centralized Version Control System*. 12

DER Diagrama de Entidade-Relacionamento. 6, 21

DVCS *Distributed Version Control System*. 12, 16

ES Engenharia de *Software*. 10, 11

FACOM Faculdade de Computação. 6, 7, 16–18, 20, 21, 24, 26, 29, 31, 33, 34

FDD *Feature-Driven Development*. 11

HTML *HyperText Markup Language*. 14

HTTP *Hypertext Transfer Protocol*. 14, 21

IoC *Inversion of Control*. 12

JSON *JavaScript Object Notation*. 21, 22

MVVM *Model-View-View-Model* . 22

NFP Número de Ficha Patrimonial. 7, 10, 18, 25

ORM *Object-Relational Mapping*. 14, 19, 21, 22

PROADI Pró-Reitoria de Administração e Infraestrutura. 7

QS Qualidade de *Software*. 11

REST *Representational State Transfer*. 19, 21, 23

SDL *Schema Definition Language*. 14

SEO *Search Engine Optimization*. 13

SoC *Separation of Concerns*. 12

SQL *Structured Query Language*. 15

SSG *Static Site Generation*. 13

SSR *Server-Side Rendering*. 13

UAS Unidade Administrativa Setorial. 7

UFMS Universidade Federal de Mato Grosso do Sul. 7, 16, 18, 24, 26, 33, 34

UI *User Interface*. 14, 22

VCS *Version Control Systems*. 12

Lista de Figuras

Capítulo 1

Introdução

A FACOM é uma Unidade Administrativa Setorial (UAS) da Universidade Federal de Mato Grosso do Sul (UFMS), responsável por cinco cursos de graduação e quatro programas de pós-graduação, atendendo atualmente a mais de 1500 estudantes e a cerca de 75 servidores, entre técnicos e docentes. Para dar suporte às suas atividades acadêmicas, a unidade mantém um amplo conjunto de bens móveis, essenciais à infraestrutura logística, administrativa e educacional.

Cada bem possui registro no Sistema de Patrimônio da UFMS, gerido pela Pró-Reitoria de Administração e Infraestrutura (PROADI). Esse registro contém dados descritivos, como Número de Ficha Patrimonial (NFP), classe, grupo, fornecedor, valor e *status*, mas não inclui informações acerca da localização física dos itens nos espaços da FACOM (salas, gabinetes, laboratórios, auditórios), nem sobre o servidor responsável pelo uso de cada item. A responsabilidade formal por todos esses bens recai sobre o ocupante do cargo Gestor da unidade, o que concentra atribuições e dificulta a gestão cotidiana.

O controle patrimonial é realizado anualmente por meio de inventário, no qual é confirmada a existência, a localização e o estado dos itens. Os bens podem estar em situação de "ativos"ou "baixados". Bens ativos podem estar em uso na unidade ou temporariamente alocados a outra unidade, denominada "posseira". Os bens baixados são aqueles comprovadamente sem condições de uso, por atestação das unidades gestoras superiores. Quando um item não é localizado durante o inventário, é classificado como "extraviado", o que implica responsabilização administrativa do gestor da unidade setorial.

1.1 Descrição do problema

Na FACOM, o acervo patrimonial ultrapassa 3000 itens distribuídos pelos prédios da unidade, sem indicação precisa de alocação. A responsabilidade formal por todo esse conjunto permanece centralizada na pessoa que ocupa a Direção da Unidade, o que intensifica os desafios na gestão.

Diante das limitações do controle manual, alguns servidores da FACOM desenvolveram, por iniciativa própria, um sistema *web* simples para registrar movimentações e complementar informações dos itens. Embora funcional, essa aplicação operava de forma isolada: por não haver sincronização com a base oficial da Universidade, com o passar do tempo, gerou-se grande quantidade de inconsistências nos registros; tampouco era possível identificar usuários como corresponsáveis pelos bens sob sua guarda. Como resultado, o nível de rastreabilidade e controle permanecia baixo.

A dinâmica constante de movimentação (mudanças de salas, reorganização de laboratórios e empréstimos temporários) amplia-se o risco de extravios ao longo dos anos. No inventário patrimonial de 2025, registraram-se 244 extravios acumulados entre 2018 e 2025, evidenciando a fragilidade do processo.

Nesse contexto, tornou-se indispensável uma solução mais robusta para gerenciar e documentar os bens da unidade, garantindo a identificação da localização dos itens, a rastreabilidade das movimentações e a atribuição de corresponsabilidade a servidores e estudantes envolvidos nos processos de empréstimo e (re)alocação patrimonial.

1.2 Objetivo

O objetivo central deste trabalho é desenvolver uma aplicação *web* moderna que possibilite o controle e gestão dos itens patrimoniais lotados na FACOM, a partir de uma base de dados sincronizada com os dados oficiais da Universidade, que permita registrar e exibir a localização exata de cada item.

1.2.1 Objetivos específicos

Com base nos requisitos funcionais e não funcionais definidos no projeto, os objetivos específicos deste trabalho são:

1. Implementar sistema de cadastro de itens patrimoniais, devendo conter o registro de sua localização exata dentro da unidade;
2. Implementar interface para visualização dos itens patrimoniais com opções de filtros;
3. Implementar interface para visualização do histórico de movimentações dos itens patrimoniais;
4. Implementar interface para visualizar os empréstimos ativos;
5. Implementar módulo de gerenciamento de movimentações de itens (empréstimos e alocações);
6. Implementar geração de termos de corresponsabilidade, em arquivos .pdf, no ato de empréstimo de itens;
7. Implementar geração de termos de corresponsabilidade, em arquivos .pdf, pelos itens contidos em determinado espaço/sala;
8. Implementar geração de relatórios de itens patrimoniais em arquivos .xlsx;
9. Promover boa experiência para o usuário através de interfaces responsivas e intuitivas;
10. Implementar arquitetura em camadas, facilitando manutenção e escalabilidade;
11. Realizar testes automatizados no *backend* e testes exploratórios no *frontend*;

A implantação do sistema não fez parte dos objetivos, visto que ainda depende da validação e autorização do setor de gestão patrimonial e Direção da FACOM, para que seja executada em servidor interno da unidade.

1.3 Resultados Esperados

Com a conclusão do desenvolvimento da aplicação *web*, espera-se estabelecer um novo padrão de controle patrimonial na Faculdade de Computação. A integração do sistema com a base oficial da Universidade não depende da UAS e sim da administração central, mas o sistema já está preparado para facilitar a carga e atualização do seu banco para manter a sincronização manual até que a universidade autorize uma sincronização automatizada.

Por meio das informações de localização dos itens, a rastreabilidade torna-se possível e isso facilitará a rotina de conferência, reduzirá o tempo gasto em inventários anuais e minimizará perdas de informação decorrentes de processos manuais.

A disponibilização de interfaces intuitivas para visualização, filtragem e consulta do histórico de movimentações tende a ampliar a transparência das ações de gestão, permitindo que os gestores conheçam com clareza a trajetória de cada item.

Um impacto particularmente relevante diz respeito à redução de extravios. A existência de registros contínuos, combinada com a obrigatoriedade de atribuição de responsável nos empréstimos e alocações, deve diminuir significativamente o número de bens não localizados durante os inventários anuais. A expectativa é de que, ao longo dos primeiros ciclos de uso do sistema, haja queda progressiva no número de extravios, tanto pelo aumento da rastreabilidade, quanto pela mudança comportamental dos usuários da unidade.

Do ponto de vista institucional, o sistema representa um avanço na profissionalização da gestão patrimonial da FACOM. Ele tende a facilitar a tomada de decisão da Direção, otimizar o trabalho dos servidores envolvidos no controle de patrimônio e oferecer maior segurança administrativa ao setor. Espera-se também que a geração de relatórios favoreça análises gerenciais, identificação de padrões de uso e planejamento de reposições.

Ademais, espera-se que a geração automática de termos de co-responsabilidade e empréstimo em formato digital, facilite processos que tradicionalmente ocorriam de maneira informal ou sem registro adequado. A médio prazo, a utilização do sistema pode estimular uma cultura de cuidado coletivo com o patrimônio público, reforçando o entendimento de que a responsabilidade pela integridade e correta alocação dos itens não deve recair exclusivamente sobre a Direção, mas deve ser distribuída de forma consciente entre servidores e estudantes.

Embora a implantação final dependa de validação institucional da UFMS e da FACOM, o sistema desenvolvido estabelece as bases técnicas e funcionais para uma gestão patrimonial mais eficiente, transparente e alinhada às necessidades contemporâneas de uma unidade acadêmica em crescimento. Em última instância, espera-se que este projeto contribua para fortalecer a governança, reduzir perdas, qualificar os processos administrativos de gestão patrimonial e apoiar a sustentabilidade do patrimônio institucional ao longo do tempo.

1.4 Organização do Trabalho

Esta monografia está organizada como segue: o Capítulo 2 apresenta conceitos que fundamentam e compõem o conteúdo e arquitetura do projeto; o Capítulo 3 detalha a metodologia aplicada ao desenvolvimento; o Capítulo 4 apresenta os resultados obtidos, destacando as funcionalidades implementadas e exibindo diagrama e imagens que ilustram o comportamento da aplicação; e por fim, o Capítulo 5 apresenta as considerações finais.

Capítulo 2

Referencial Teórico

Neste capítulo são apresentados os conceitos fundamentais que sustentam a arquitetura e o desenvolvimento do projeto. São discutidos princípios de Engenharia de *Software*, bem como as metodologias, tecnologias e ferramentas empregadas na construção de sistemas *web*, estabelecendo a base teórica necessária para compreender as escolhas de projeto adotadas.

2.1 Conceitos e abstrações do projeto

- **Patrimônio:** patrimônio público é o conjunto de bens e direitos, tangíveis e intangíveis, que pertencem ao Estado e são administrados para atender às necessidades coletivas [2]. No contexto deste projeto, entende-se por patrimônio todos os bens móveis cedidos à FACOM, pela UFMS, que vão compor o acervo material necessário para o exercício de toda a atividade administrativa e educacional da faculdade. Alguns exemplos são: armários, cadeiras, mesas, computadores, *notebooks*, projetores, etc. Cada item patrimonial possui uma identificação única denominada NFP;

- **Alocação:** no contexto do projeto, alocação refere-se a toda ação de registro ou mudança de localização de um item. Isto é, para novos itens que chegam na FACOM, deve ser feito o seu cadastro no sistema com alocação de sala. Para itens já existentes na unidade, é possível fazer realocações mudando suas localizações (local/sala) no sistema.

- **Empréstimo**: no âmbito do projeto, empréstimo refere-se a toda ação de ceder o uso de itens patrimoniais, temporariamente, a servidores ou estudantes. Em cada ato, é necessário a geração de um termo de responsabilidade pelo item.

2.2 Engenharia de *Software* e Metodologias ágeis

A Engenharia de *Software* (ES) é uma área da Computação que aplica abordagens de engenharia ao desenvolvimento de *software*, preocupando-se com todos os aspectos, desde sua produção, operação e manutenção [3]. O principal objetivo é garantir a produção de sistemas confiáveis, eficientes e de alta qualidade, dentro de prazos e orçamentos estabelecidos. Para isso, a área abrange um extenso conjunto de ferramentas, metodologias e procedimentos que visam gerenciar a complexidade do desenvolvimento.

Tradicionalmente, os processos de ES eram marcados por modelos preditivos e sequenciais, como o Modelo Cascata, que exigem o planejamento completo do projeto antes de iniciar a codificação [4]. Se por um lado esses modelos oferecem controle rigoroso sobre o projeto, por outro, demonstram pouca flexibilidade para lidar com requisitos mutáveis, o que se revelou um grande desafio à natureza dinâmica dos projetos de *software*.

Foi diante disso que as metodologias ágeis ganharam grande espaço no início dos anos 2000, por favorecerem ciclos curtos de entrega, revisão contínua e adaptação rápida a mudanças. Enquanto a ES fornece o conjunto de técnicas para a construção (como análise de requisitos, *design* de arquitetura e testes), as metodologias ágeis fornecem os modelos de processo e gerenciamento do projeto, para aplicar essas técnicas de forma eficiente e flexível [3].

2.2.1 Desenvolvimento orientado por funcionalidades

O *Feature-Driven Development* (FDD) é uma metodologia ágil e iterativa de desenvolvimento de *software* que se concentra na entrega rápida e incremental de funcionalidades [5]. O principal objetivo do FDD é manter a **rastreabilidade** e garantir a **entrega contínua de valor**, mediante **priorização** de demandas, organizando todo o ciclo de vida do projeto em torno do desenvolvimento dessas funcionalidades.

A escolha de aplicar o FDD ao projeto objeto deste trabalho, de maneira simplificada, foi devido à facilidade na elaboração da lista de requisitos e funcionalidades, e na priorização das entregas de acordo com as funcionalidades de maior valor.

2.2.2 Qualidade de *Software*

A Qualidade de *Software* (QS) abrange um conjunto de características que garantem que o produto final atenda aos requisitos explícitos e implícitos do usuário e da organização. A qualidade abrange a conformidade com os requisitos funcionais e de desempenho especificados, assim como com padrões de desenvolvimento bem documentados [4].

O objetivo da QS é garantir que o produto seja o mais livre de defeitos possível, e que atenda ao valor esperado pelo cliente. No âmbito do desenvolvimento de *software* moderno, essa garantia é amplamente delegada à atividade de testes, que normalmente é planejada e executada de forma estratégica. Para tanto, é bastante comum que projetos ágeis apliquem o modelo conceitual da **Pirâmide de Testes**. Seu modelo original sugere que a maioria dos testes deve estar na base da pirâmide, onde são mais baratos e rápidos de executar, diminuindo em volume à medida que sobem na hierarquia [6].

Os **testes unitários** estão na base da pirâmide, sendo os mais numerosos, rápidos, baratos de implementar e altamente automatizados, com foco em testar a menor unidade de código isoladamente (métodos, classes e/ou funções). No meio da pirâmide estão os **testes de integração**, que têm por objetivo validar a comunicação correta entre diferentes módulos, serviços ou componentes, por exemplo, testes de integração com bancos de dados, APIs ou serviços externos. No topo, encontram-se os **testes de UI** (*User Interface*), envolvendo fluxos de testes *end-to-end* na interface gráfica do usuário. Eles simulam o comportamento do usuário final e buscam validar a sua experiência. No entanto, são os mais lentos, os mais caros e os mais sujeitos a quebras por pequenas mudanças na UI.

Outra modalidade de testes é a de **testes exploratórios**, método manual em que o testador explora o sistema sem casos de teste pré-definidos, utilizando sua criatividade e conhecimento para descobrir falhas inesperadas ou *bugs* (defeitos) que a automação pode

não cobrir[6].

Em suma, todas as atividades de testes constituem uma estratégia de Qualidade de *Software* com o objetivo de garantir um ciclo de *feedback* rápido ao desenvolvedor, reduzir o custo de correção de defeitos e, portanto, reforçar a confiança na base de código, formando um pilar essencial do desenvolvimento *software* moderno.

2.3 Versionamento e Sistemas de Controle de Versões

Versionamento, ou controle de versão, é um processo importante na área da Engenharia de *Software* que consiste em registrar as alterações feitas em um arquivo ou em um conjunto de arquivos ao longo do tempo[7]. Sua principal função é permitir que desenvolvedores rastreiem e recuperem versões específicas do projeto a qualquer momento, garantindo a rastreabilidade e a integridade do código-fonte.

Nesse sentido, o uso de Sistemas de Controle de Versão - *Version Control Systems* (VCS) - é fundamental para mitigar riscos de desenvolvimento, como a perda acidental de código, e para gerenciar a complexidade inerente ao trabalho colaborativo no desenvolvimento de *software*[7].

Tais ferramentas implementam o princípio da Separação de Preocupações - *Separation of Concerns* (SoC) - no fluxo de trabalho, de modo que isolam o processo de rastreamento de mudanças da lógica de negócio. Isto é, elas possibilitam que o código-fonte se mantenha focado na sua finalidade lógica, enquanto o sistema de versionamento cuida de toda a complexidade de gerenciar a evolução e as atualizações do projeto.

O **Git** é o Sistema de Controle de Versão Distribuído - *Distributed Version Control System* (DVCS) - mais utilizado no desenvolvimento de *software*. Desenvolvido por Linus Torvalds, ele se diferencia de sistemas centralizados - *Centralized Version Control System* (CVCS) - por sua arquitetura. O controle é dito distribuído, de modo que diversos servidores podem ter uma cópia completa do repositório e, caso algum servidor deixe de funcionar, o projeto poderá ser restabelecido a partir de qualquer um dos demais repositórios.

Já o **GitHub** é essencialmente uma plataforma *web* que oferece serviço de hospedagem de repositórios, centralizando a interação entre as cópias distribuídas dos desenvolvedores por meio de comandos de envio (*push*) e recebimento (*pull*). Ele facilita a prática de revisão de código e a colaboração, tendo introduzido e padronizado o modelo de *Pull Request* para o desenvolvimento distribuído, tornando-se um componente crítico do ecossistema de código aberto e do desenvolvimento profissional [8].

2.4 *Frameworks*

Um *Framework* pode ser definido como um conjunto de bibliotecas, ferramentas e diretrizes que fornece uma estrutura semiacabada para o desenvolvimento de aplicações. Ele não é apenas uma biblioteca. Enquanto uma biblioteca oferece funções que o desenvolvedor reutiliza, um *framework* impõe um controle de fluxo ao desenvolvimento, chamando o código do desenvolvedor em momentos específicos, seguindo o princípio da Inversão de Controle - *Inversion of Control* (IoC)[9]. O principal objetivo de um *framework* é aumentar a produtividade do desenvolvedor e a reutilização de código, fornecendo uma base testada, documentada e padronizada para resolver problemas comuns de um domínio específico.

A conceituação moderna de *framework*, no contexto da *stack* JavaScript, vai além da IoC para enfatizar a **componentização** e a **otimização** da experiência de desenvolvimento na criação de interfaces de usuário complexas e de alto desempenho.

Assim, pode-se dizer que um *framework* JavaScript moderno é uma plataforma completa e arquiteturalmente prescritiva que fornece mecanismos para a componentização modular de interface, o gerenciamento reativo de estados e soluções otimizadas de renderização - como *Server-Side Rendering* (SSR) ou *Static Site Generation* (SSG) -, com o objetivo de garantir alta performance e manutenibilidade em escala[10, 11]. Ele encapsula o desenvolvimento *full-stack* e impõe convenções para reduzir a carga cognitiva do desenvolvedor.

2.4.1 Nuxt

Nuxt é um *framework* de desenvolvimento *web fullstack*, de código aberto, construído sobre o ecossistema VueJS (*client-side*), estendendo sua funcionalidade para a camada de servidor (*server-side*)[10]. Seu principal propósito é simplificar a criação de aplicações VueJS, fornecendo uma estrutura padronizada para as camadas *frontend* e *backend*, configurações pré-definidas e convenções robustas para renderização. As definições e conceitos-chave a respeito desse poderoso *framework* giram em torno de três pilares:

1. **Arquitetura Baseada em Convenção:** o Nuxt implementa rigorosamente o princípio da Convenção sobre Configuração - *Convention over configuration* (CoC) -, na medida em que, em vez de exigir que o desenvolvedor configure ferramentas complexas como, por exemplo, roteamento, ele infere as funcionalidades do aplicativo a partir da estrutura de diretórios e nomes de arquivos. A aplicação deste princípio visa aumentar a produtividade e a manutenibilidade do código[12].

2. **Renderização Híbrida e Otimização:** um dos aspectos mais valorizados do Nuxt é o suporte nativo a múltiplos modos de renderização (SSR, SSG). São recursos fundamentais para otimizar o desempenho, a experiência do usuário e a otimização para motores de busca - *Search Engine Optimization* (SEO) - em aplicações *web* modernas[13].

3. **Abstração de Ferramentas (Zero-Config):** o Nuxt abstrai grande parte da complexidade de configurações de um *framework*. Ele integra e configura automaticamente ferramentas essenciais da stack JavaScript, como: toda a arquitetura VueJS para a construção de UI, o *bundler* Vite para empacotar código, o suporte nativo para tipagem Typescript[10]. Tais abstrações permite que o desenvolvedor se concentre mais nas lógicas de negócio e menos na infraestrutura de *build*.

2.5 Bibliotecas

No contexto do desenvolvimento de *software*, uma "biblioteca"é uma coleção de rotinas, funções e classes pré-escritas que fornecem serviços úteis a outras partes de um programa[4]. Diferente de um *framework*, uma biblioteca expõe ao desenvolvedor um conjunto de ferramentas que ele pode utilizar ativamente para realizar tarefas específicas, sem que isso defina previamente a arquitetura ou o fluxo de controle central da aplicação.

Em outras palavras, são implementações de algoritmos complexos para serem reutilizados em diversas partes de um código-fonte, e inclusive, em diversos projetos. Alguns

exemplos são: bibliotecas para manipulação de datas, de chamadas via *Hypertext Transfer Protocol* (HTTP), ou até mesmo para renderização de *User Interface* (UI).

2.5.1 NuxtUI

NuxtUI é uma biblioteca de interface de usuário de código aberto, especificamente projetada para o ecossistema Nuxt e VueJS, que oferece uma coleção de componentes totalmente responsivos, estilizados e acessíveis[14]. É uma ferramenta que materializa o conceito de *Component-Driven Development* (CDD)[13], que preconiza a construção de interfaces de usuário a partir de módulos isolados e reutilizáveis, os famosos **componentes**.

2.5.2 TailwindCSS

TailwindCSS é uma biblioteca *utility-first* de *Cascading Style Sheets* (CSS) que fornece um extenso conjunto de classes de utilidade de baixo nível, permitindo que os desenvolvedores implementem *designs* personalizados diretamente em seus arquivos de marcação - *HyperText Markup Language* (HTML) ou componentes - sem a necessidade de escrever código CSS tradicional[15].

O *design utility-first* é característica central do Tailwind: ao invés de classes semânticas tradicionais que abrigam uma ou mais propriedades CSS, ele fornece classes que representam uma única propriedade. O principal objetivo desse modelo é eliminar o custo de tempo e produtividade de alternar entre os arquivos de marcação e de estilo, permitindo que o desenvolvedor construa interfaces de usuário complexas combinando classes atômicas e imutáveis.

2.5.3 Prisma ORM

O Prisma é um *toolkit* de código aberto que atua como um moderno *Object-Relational Mapping* (ORM) para NodeJS e TypeScript. Ele é caracterizado por uma arquitetura em três partes, focada em segurança de tipo (*type safety*) e na produtividade do desenvolvedor [16]. O Prisma é classificado como biblioteca porque fornece um robusto conjunto de ferramentas dirigidas à persistência de dados e ao acesso a banco de dados, sem ditar a arquitetura geral da aplicação *backend*.

Como mencionado, sua arquitetura é composta por três principais núcleos que trabalham de maneira integrada para otimizar o fluxo de trabalho sobre acesso e manipulação de banco de dados:

1. **Prisma *Schema*:** consiste em um arquivo de configuração central, escrito em Linguagem de Definição de Esquema - *Schema Definition Language* (SDL) - que define o modelo de dados da aplicação, tais como as entidades do banco, seus atributos, tipos e os relacionamentos;

2. **Prisma *Migrate*:** gerencia o esquema do banco de dados, utilizando-se do arquivo *schema* para gerar e aplicar *migrations* SQL, garantindo que o esquema do banco esteja sempre sincronizado com o modelo de dados definido para a aplicação;

3. **Prisma *Client*:** consiste em um *query builder* (gerador de consultas) automático a partir do modelo defindo pelo arquivo *schema*. Ele oferece uma *Application Programming Interface* (API) *type-safe* (TypeScript) em tempo de compilação para

interagir com o banco de dados. A geração automática de um cliente *type-safe* elimina a necessidade de mapeamento manual e garante que as operações de banco de dados sejam verificadas por erros de tipo em tempo de desenvolvimento [16].

O Prisma promove uma abstração de alto nível das operações CRUD (*create, read, update, delete*) manuais em *Structured Query Language* (SQL) por métodos JavaScript/TypeScript fáceis de usar. Essa abordagem tem ganhado popularidade no ecossistema JavaScript por oferecer uma camada mais limpa de acesso a dados, permitindo que desenvolvedores apliquem padrões arquiteturais de forma mais eficiente.

2.6 Considerações Finais

Este capítulo apresentou os principais conceitos, ferramentas e tecnologias adotadas no projeto. A forma como foram integrados durante o desenvolvimento do sistema está descrita no Capítulo 3.

Capítulo 3

Metodologia

Este capítulo apresenta a metodologia adotada no desenvolvimento do sistema de gestão interna de patrimônios da FACOM/UFMS. São descritos o planejamento do projeto, a abordagem arquitetural escolhida e as estratégias empregadas para garantir a qualidade do *software*.

A demanda pelo sistema foi formalizada pela Coordenação Administrativa e pela Direção da FACOM, e a solução foi concebida e desenvolvida no âmbito deste Trabalho de Conclusão de Curso. Trata-se de uma pesquisa aplicada, de natureza tecnológica, orientada à criação de um *software* voltado à solução de um problema concreto de gestão patrimonial na unidade. O trabalho fundamenta-se na aplicação prática de técnicas e metodologias de Engenharia de *Software*, para atender à necessidade institucional.

3.1 Materiais e métodos

3.1.1 Reuniões presenciais e *Google meet*

Foram realizadas reuniões na FACOM com os servidores interessados no desenvolvimento do sistema, no intuito de discutir e definir os requisitos funcionais da aplicação, bem como esclarecer dúvidas. O *Google Meet* também foi utilizado como meio de comunicação para realizar reuniões virtuais.

3.1.2 Editor de código-fonte

O *Visual Studio Code* (VSCode) foi utilizado como ferramenta de desenvolvimento do código-fonte do sistema. No *workspace* do projeto, foram configuradas ferramentas de formatação e padronização, como *Prettier*[17] e *ESlint*[18], a fim de manter a qualidade do código.

3.1.3 Sistema de controle de versões

Foi utilizado o Git como sistema de controle de versão distribuído (DVCS) para manter a segurança e integridade do código-fonte, cujo repositório remoto foi hospedado na plataforma Github, a fim de facilitar seu acesso por outros desenvolvedores que venham futuramente dar continuidade ao projeto[19].

3.2 Desenvolvimento incremental

Por iniciativa de servidores da FACOM, foi concebida inicialmente uma aplicação *web* simples na tentativa de facilitar e automatizar a rotina de gestão do patrimônio da unidade. Com ele, foi possível manter o registro dos bens e suas localizações, mas no decorrer do tempo surgiram demandas mais complexas do que ele poderia lidar, trazendo à tona a necessidade de implementar um sistema mais robusto e moderno, especialmente melhorias da base de dados e das tecnologias de desenvolvimento e usabilidade.

Em contrapartida, ele foi de suma importância para orientar a definição dos requisitos do sistema objeto de trabalho, pois a nova aplicação deveria abranger as funcionalidades já existentes e incluir novas demandas do setor de patrimônios da FACOM.

3.2.1 Planejamento e análise de requisitos

Para melhor definição dos requisitos do projeto, procedeu-se à análise do sistema pré-existente para identificar suas funcionalidades e mapeá-las como requisitos, com as devidas melhorias, para o novo projeto. Nesta etapa, foram essenciais alguns *brainstorms* com servidores e com a direção da FACOM, pessoas diretamente envolvidas e responsáveis pela gestão patrimonial da unidade, para compreender as necessidades e as novas demandas desse gerenciamento.

Uma vez entendidos e definidos os requisitos funcionais do projeto, seguiu-se ao mapeamento das funcionalidades. Devido ao curto prazo disponível para desenvolvimento, foi necessário organizá-las em ordem de prioridade, selecionando quais integrariam a primeira versão entregável do sistema e quais seriam entregues posteriormente em uma segunda versão.

Nesse sentido, verificou-se que a prioridade envolvia a implementação de dois módulos: o de **gerenciamento do ciclo de vida dos bens patrimoniais**; e o de **gerenciamento de suas movimentações**, referentes às ações de empréstimos e (re)alocações. Em primeira mão, esses módulos foram avaliados como o *core* funcional da aplicação.

Quanto à população da base de dados, optou-se por fazê-la via *seeds scripts*, que operam a partir da leitura de arquivos *Comma-Separated Values* (CSV) contendo os dados necessários. Em particular, os dados dos itens patrimoniais foram obtidos do relatório anual de inventário patrimonial de 2025, armazenado no sistema de patrimônio da universidade.

As novas demandas, consideradas essenciais, relacionam-se à necessidade de que o sistema gere termos de corresponsabilidade dos bens patrimoniais. Estes termos fazem parte do processo de movimentações por razão de empréstimos ou da responsabilização pelo uso de itens por parte dos servidores técnicos e docentes, no intuito de aumentar o controle e engajamento dos responsáveis na gestão e zelo patrimonial.

Existem, ainda, outras funcionalidades com capacidade de agregar valor ao projeto, como o módulo de gerenciamento de usuários e permissões. Estas funcionalidades foram, em princípio, definidas como não essenciais e, então, mapeadas para uma segunda versão do sistema.

3.2.2 Divisão de funcionalidades e pacotes entregáveis do projeto

Abaixo estão elencadas as funcionalidades que agregam valor ao projeto, tendo sido selecionadas aquelas essenciais para compor o objeto deste trabalho e o primeiro pacote

entregável (versão 1.0) da aplicação. Já as funcionalidades não essenciais foram mapeadas para o segundo pacote entregável (versão 2.0).

Primeiro pacote entregável:

- **Gerenciamento de Itens:**
  - importação dos itens registrados na base de dados oficial da UFMS, lotados na FACOM;
  - cadastrar novo item;
  - listar itens com opção de filtro por situação interna;
  - busca de itens por NFP, descrição, classe ou grupo;
  - geração de relatórios em arquivos CSV, baseada na situação interna do item;
  - interface para consultar todos os dados salvos na base relativos aos itens;
  - interface para consultar o histórico de movimentação dos itens.
- **Gerenciamento de Movimentações:**
  - cadastrar movimentação de item, podendo ser do tipo "alocação"ou "empréstimo";
  - listar movimentações;
  - geração do termo de compromisso no cadastro de empréstimo para estudantes;
  - geração do termo de compromisso no cadastro de empréstimo para servidores;
  - geração de relatórios de empréstimos em arquivos CSV;
- **Gerenciamento de Locais:**
  - cadastrar os locais no banco, via *seed script*, a partir do arquivo CSV fornecido pela Direção da FACOM;
  - listar locais;
- **Gerenciamento de Salas:**
  - cadastrar as salas no banco, via *seed script*, a partir do arquivo CSV fornecido pela Direção da FACOM;
  - listar salas;
  - geração do termo de corresponsabilidade pelos itens lotados em salas;
- **Gerenciamento de Responsáveis:**
  - cadastrar os responsáveis no banco, via *seed script*, a partir do arquivo CSV fornecido pela Direção da FACOM;
  - listar responsáveis;

Segundo pacote entregável:

- **Gerenciamento de Locais:**
  - cadastro de locais via *frontend*;
  - edição e deleção de locais;
- **Gerenciamento de Salas:**
  - cadastro de salas via *frontend*;
  - edição e deleção de salas;
- **Gerenciamento de Responsáveis:**
  - cadastro de responsáveis via *frontend*;
  - edição e deleção de responsáveis;
- **Gerenciamento de usuários e permissões:**
  - cadastrar, listar, editar, deletar;
  - recuperar senha, *login* e *logout*;

3.3 Arquitetura da aplicação

A aplicação possui uma arquitetura moderna construída com Nuxt, um *framework* que oferece estrutura *client-server* (*full stack*), subdividida em módulos, que neste projeto foram organizados da seguinte forma:

1. Camada *Frontend*: aplicação em Nuxt/Typescript que implementa as interfaces do usuário utilizando a biblioteca de componentes NuxtUI;
2. Camada *Backend*:
   (a) API *Representational State Transfer* (REST) em NodeJS/Typescript, que implementa os *endpoints* necessários para lidar com lógicas de negócio e interação com a base de dados;
   (b) Camada de Dados: implementa repositórios que utilizam Prisma ORM para persistência e acesso ao banco de dados relacional MySQL.

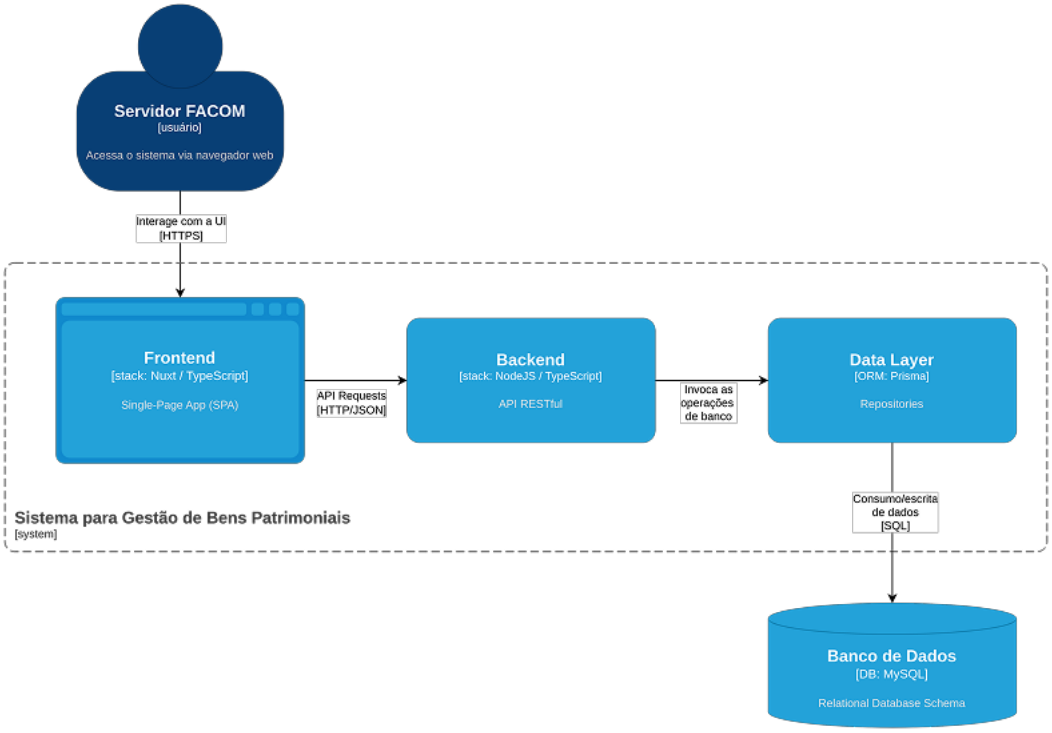

Figura 3.1: Diagrama C4 da solução base do projeto.

3.3.1 Banco de dados

Em vista dos principais modelos de banco de dados utilizados no mercado de desenvolvimento de *software*, foi escolhido para o sistema em questão o **banco de dados relacional** MySQL. Tal escolha visa valorizar a integridade dos dados e tirar melhor proveito dos princípios ACID (Atomicidade, Consistência, Isolamento e Durabilidade) propostos pelos bancos relacionais.

O banco foi projetado para atender às necessidades da aplicação, como armazenar informações sobre itens patrimoniais, suas localizações, movimentações e o pessoal responsável, possibilitando rastrear e gerenciar o ciclo de vida completo desses bens na FACOM. As principais entidades do sistema são traduzidas pelos seguintes recursos:

- **Itens**: bens físicos com propriedades como descrição, *status*, localização, classe, grupo, etc.;
- **Grupos**: categorização específica para itens;
- **Classes**: categorização genérica para itens;
- **Movimentação**: registros de transferências de itens entre locais ou empréstimos de itens a terceiros;
- **Responsável**: pessoas responsáveis por itens lotados em locais/salas específicas, ou em registro de empréstimo. Podem ser alunos, professores, ou servidores em geral;
- **Locais**: espaços físicos que agrupam as salas;
- **Salas**: espaços físicos, dentro de locais, onde os itens estão alocados.

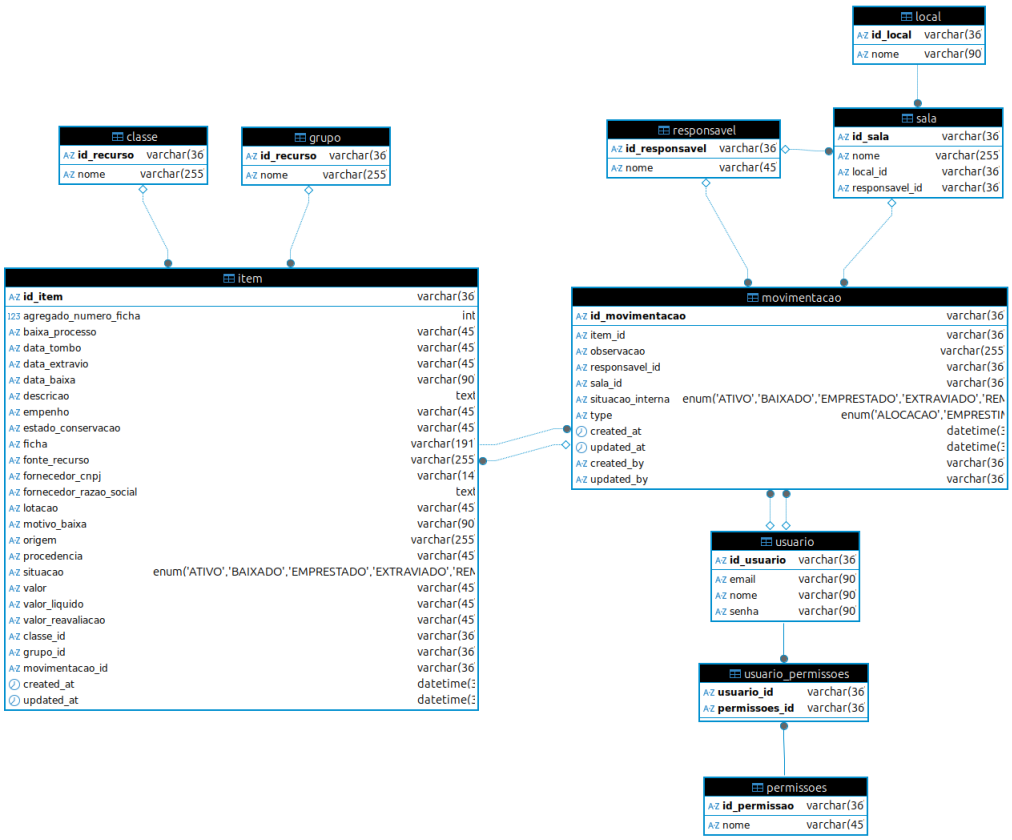

Figura 3.2: Diagrama de Entidade-Relacionamento (DER) do Sistema de Patrimônio da FACOM.

O sistema explora, ainda, o mecanismo de *Migrations* do Prisma ORM para a criação e a edição do esquema do banco de dados. A partir do Prisma *schema* é possível criar toda a estrutura inicial de tabelas e relacionamentos.

Para popular o banco com os dados, adotou-se duas estratégias. A primeira foi a importação de dados a partir de um *script* que lê o relatório anual de inventário da Universidade (em arquivo CSV), processa os dados e popula a tabela "Item"com todos os dados relativos aos bens patrimoniais. A segunda foi a implementação de *seed scripts* para preencher os demais recursos (classes, grupos, locais, salas e responsáveis).

Os detalhes de conexão com o banco de dados são gerenciados através de variáveis de ambiente e do Prisma *client*, cuja instanciação expõe os recursos necessários para que a conexão seja estabelecida.

3.3.2 Backend

Para a exposição e a escrita de dados na base, o projeto implementa uma API REST com padrão de comunicação definido pelo protocolo HTTP. Na camada de entrada de dados, os dados são validados de acordo com as regras de cada *endpoint*, e a resposta fornecida pela API segue o formato padrão *JavaScript Object Notation* (JSON).

A arquitetura do *backend* é guiada por diversos princípios fundamentais de engenharia de *software* para garantir a manutenibilidade, a testabilidade e a escalabilidade. O

projeto implementa uma clássica *Layered Architecture*[12], onde cada camada da API possui suas próprias responsabilidades, bem definidas e independentes entre si. Podem ser representadas da seguinte forma:

Tabela 3.1: Estrutura de Camadas Lógicas e Responsabilidades no Backend.

Camada	**Responsabilidade**
API Layer	Gerencia o ciclo de vida das requisições HTTP e é responsável pela formatação das respostas (JSON) ao cliente.
Service Layer	Implementa toda a lógica de negócio do sistema, coordena as operações e gerencia as transações.
Repository Layer	Abstrai e encapsula a lógica de acesso, leitura e escrita de dados, expondo métodos amigáveis à *Service Layer*.
Data Layer	Realiza a conexão física e o mapeamento com o banco de dados relacional MySQL via Prisma ORM.

Internamente, a arquitetura é formalizada por meio de alguns *design patterns*. O *Repository Pattern* encapsula toda a lógica de acesso aos dados, a abstração do Prisma ORM e a centralização das consultas. Acima dele, o *Service Layer Pattern* cuida de todas as lógicas de negócio e processamento de dados[12].

3.3.3 Frontend

A arquitetura do *Frontend* é apoiada na estrutura do Nuxt e adota o padrão baseado em Componentes com forte aderência ao modelo *Model-View-View-Model* (MVVM), uma característica intrínseca do ecossistema Vue[13].

A aplicação opera em uma *Single Page Application* (SPA), utilizando o esquema de roteamento automático *client-side* do Nuxt para proporcionar uma experiência de usuário fluida e um carregamento dinâmico de conteúdo.

O *design system* adota uma estrutura hierarquizada de componentes reutilizáveis, traduzindo uma versão simplificada do *Atomic Design*[1]:

- **Pages** atuam como Organismos, orquestrando a tela;
- **Components** atuam como Moléculas, agrupando lógica e UI;
- **UI Elements** atuam como Átomos, sendo os elementos básicos da interface.

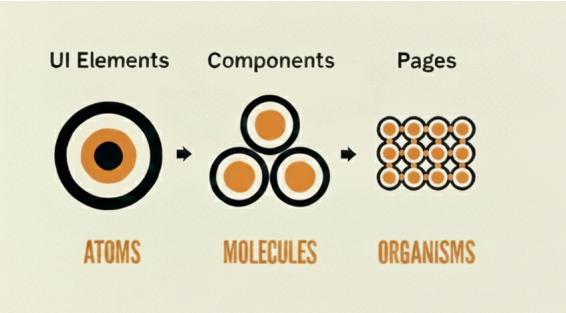

Figura 3.3: Imagem adaptada do modelo *Atomic Design*[1] aplicado ao projeto.

Quanto à comunicação entre componentes e gestão de estados, o projeto adota dois principais padrões:

- **Fluxo de dados unidirecional** (*Unidirectional Data Flow*): a comunicação entre componentes é mantida por meio da regra estrita de *props down, events up*. Ou seja, um componente pai passa dados para o filho via *props*, e o filho notifica o pai sobre eventos através de *emits*, garantindo a previsibilidade do fluxo de dados.

- **Gerenciamento de estado centralizado**: para dados globais, o projeto utiliza a biblioteca Pinia, que fornece um estado centralizado, suporte nativo a *devtools* e tipagem completa em TypeScript, permitindo que a aplicação gerencie estados complexos de forma organizada.

A comunicação com o *backend* adota o padrão de consumo de APIs REST, seguindo o protocolo HTTP. Todas as lógicas de comunicação com a API estão concentradas na camada dos *services*.

O *frontend* também adota interfaces em TypeScript que definem estritamente a estrutura dos dados de entrada e de saída, garantindo uma tipagem forte. Esse uso extensivo de tipagem permite a detecção precoce de erros em tempo de desenvolvimento, aumentando a **segurança e qualidade do código**.

A **arquitetura de roteamento** adota o modelo de *File-Based Routing* do Nuxt, ou seja, a estrutura de pastas (*pages*/**) mapeia diretamente as rotas da aplicação *frontend* e suporta rotas aninhadas para organizar a navegação de forma hierárquica e clara.

3.4 Garantia de Qualidade

A fim de garantir qualidade significativa ao sistema desenvolvido, foram implementados **testes unitários** no *backend*, utilizando a biblioteca Jest, para os principais endpoints que constituem o núcleo do sistema, validando as regras de negócio referentes ao gerenciamento dos recursos de Item e de Movimentação.

No *frontend*, devido à limitação de tempo e à priorização de demandas, decidiu-se realizar **testes exploratórios**, concentrados na comunicação com o *backend* em todos os fluxos implementados. Tal estratégia possibilitou validar a integração cliente-servidor, detectar eventuais falhas e, então, efetuar a correção.

Capítulo 4

Resultados

Este capítulo apresenta os resultados obtidos pelo desenvolvimento do sistema interno de gestão de patrimônio da FACOM, destacando as funcionalidades implementadas e a primeira versão entregue.

4.1 Visão geral do sistema

O sistema de gestão de bens patrimoniais da FACOM/UFMS foi desenvolvido como uma aplicação *web* funcional, capaz de estabelecer um novo padrão de controle patrimonial e estimular a cultura de cuidado coletivo sobre o patrimônio público.

A aplicação atende ao objetivo principal almejado: armazenar e exibir as informações oficiais sobre os itens patrimoniais, especialmente a localização exata deles dentro da unidade, possibilitar efetiva gestão das movimentações (empréstimos e alocações) desses bens, além de permitir formalizar termos de corresponsabilidade para servidores e estudantes pelo uso de bens.

O sistema apresenta, de modo geral, interfaces de usuário intuitivas e de fácil navegação, oferece filtragem de dados, consulta do histórico de movimentações, consulta de empréstimos ativos, geração de relatórios e de termos de corresponsabilidade, facilitando e tornando mais confiáveis os processos de gestão patrimonial da unidade.

A página inicial do sistema, ilustrada na Figura 4.1, apresenta dados quantitativos acerca dos itens patrimoniais lotados na unidade, bem como oferece o acesso rápido a principais funcionalidades como cadastro e consulta de itens, cadastro e consulta de empréstimo e geração de relatórios.

Figura 4.1: Página inicial da aplicação.

A arquitetura técnica proposta foi consolidada por meio do emprego de ferramentas e tecnologias modernas capazes de garantir testabilidade, manutenibilidade e escalabilidade do sistema.

4.2 Módulo de gerenciamento de itens patrimoniais

Este módulo faz parte do núcleo principal da aplicação e reúne as regras relativas aos fluxos que envolvem o ciclo de vida dos itens patrimoniais. Na interface de listagem dos itens, o usuário pode visualizar, filtrar e buscar itens patrimoniais pelo NFP, pela descrição, pela classe ou pelo grupo (Figura 4.2).

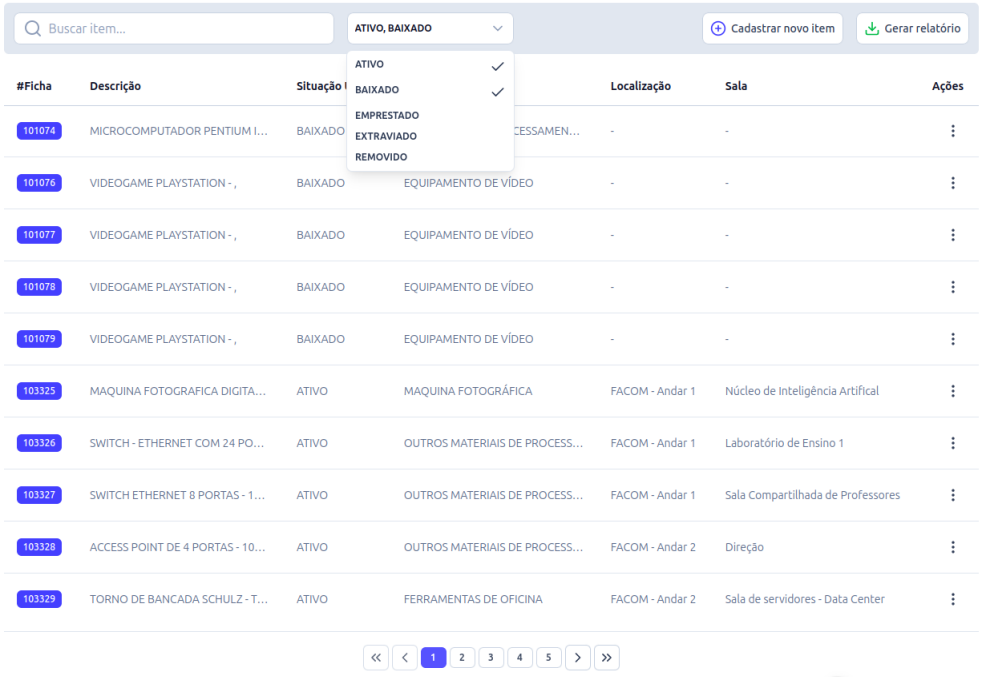

Figura 4.2: Tabela paginada para listagem de itens.

De acordo com os dados oficiais da Universidade, os itens possuem dois estados válidos: **"ativo"** ou **"baixado"**. Os ativos são todos aqueles que se encontram em pleno uso e circulação. Os baixados são aqueles que, por qualquer razão, deixaram de integrar o patrimônio da Universidade. Para a UFMS, os itens extraviados são considerados ativos, mas possuem uma "data de extravio"em seus registros para identificá-los como tais. Mas, para fins de **controle interno**, adotaram-se outros três estados: **"extraviado"**, **"emprestado"** ou **"removido"**, dos quais o último indica que o item **saiu da lotação FACOM** por qualquer motivo, mas permanecerá na base de dados com esse *status* para manter seu histórico.

Os referidos *status* constituem filtros para a listagem dos itens, podendo também ser aplicados na emissão de relatórios que favorecem análises gerenciais, identificação de padrões de uso e o planejamento de reposições de itens.

O sistema também permite o cadastro de novos itens, como mostrado na Figura 4.3. Em conjunto com esta ação, é registrada também uma movimentação inicial de "alocação"para determinar a localização do item. Nesse sentido, a modelagem do banco de dados foi fundamental para definir o relacionamento Item-Movimentação, de modo que qualquer item registrado sempre terá uma "movimentação atual"vinculada, indicando se ele está em um espaço físico da unidade ou emprestado a terceiros, a fim de garantir sua rastreabilidade.

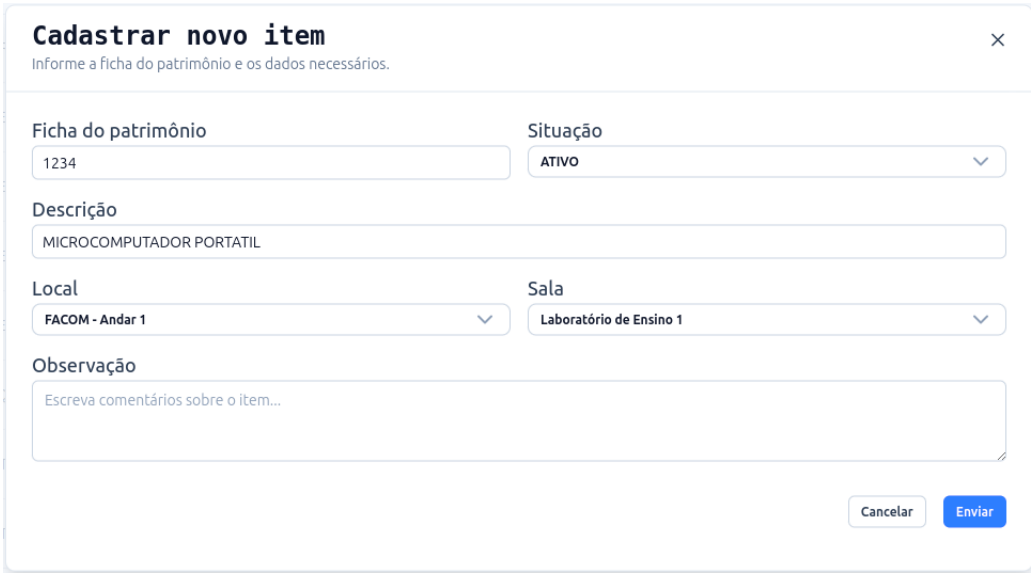

Figura 4.3: Formulário para o cadastro de um novo item patrimonial.

A partir da exibição de cada item, é possível visualizar os "detalhes"e o "histórico"do item. Os detalhes contêm as demais informações complementares provenientes da base de dados, como, por exemplo, datas de baixa ou de extravio, quando houver, fornecedores, valores, etc. Também é possível visualizar o histórico completo das movimentações ocorridas durante o ciclo de vida do item na sua lotação. Tais funcionalidades ampliam a transparência dos processos de gestão patrimonial aos usuários.

4.3 Módulo de gerenciamento de movimentações

Este módulo, juntamente com o anterior, compõe o núcleo principal do sistema, abrangendo todas as regras das movimentações sobre os itens patrimoniais. Foram definidos dois tipos de movimentação: **alocação** (mudança de localização) e **empréstimo** a servidores ou estudantes. Para compreender as regras de ambos os processos, a Figura 4.4 detalha o processo de registro de movimentações.

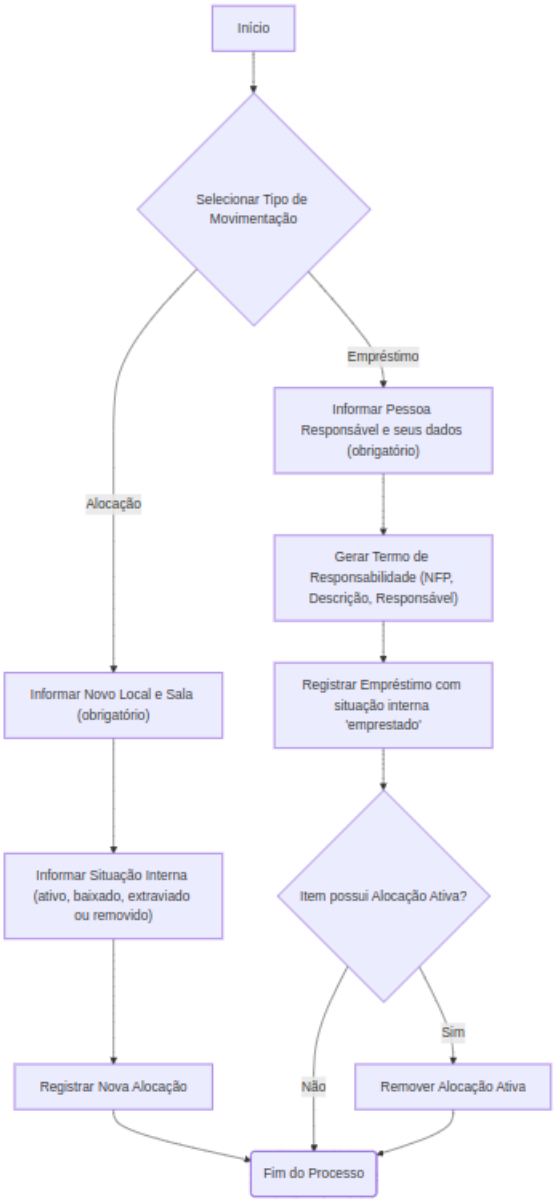

Figura 4.4: Diagrama do processo de registro de movimentações.

Em casos de (re)alocação, por se tratar da ação que define a localidade do item, é imprescindível que o usuário informe o novo Local e Sala, para manter a rastreabilidade (Figura 4.5).

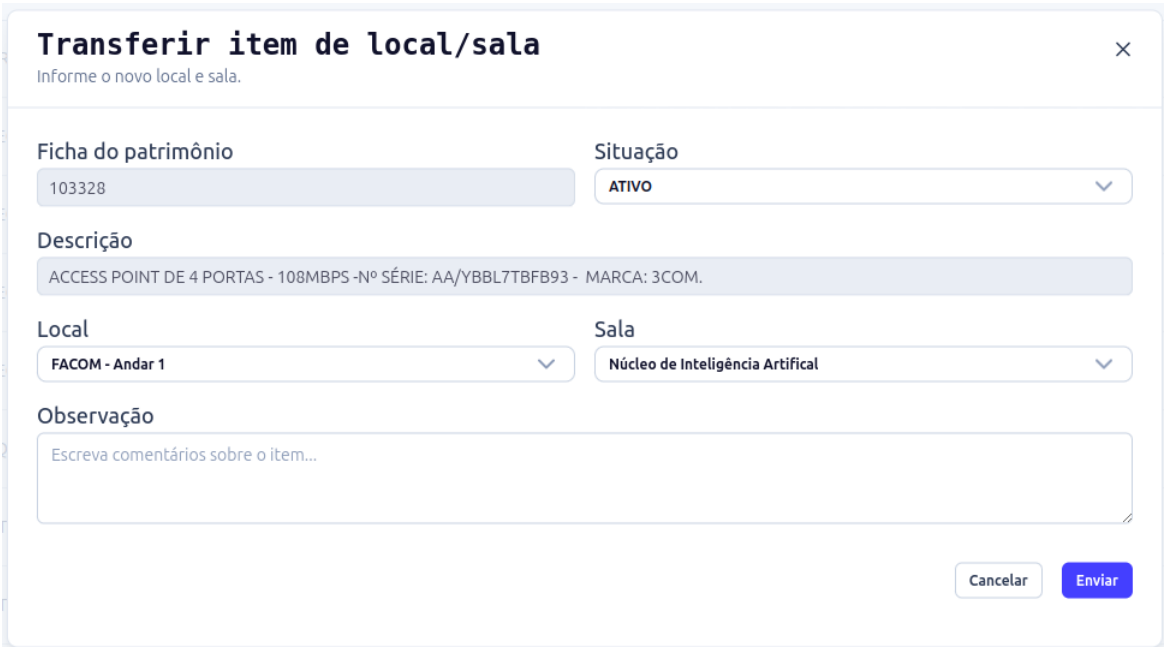

Figura 4.5: Formulário para registro de alocação de item patrimonial.

Entende-se por empréstimo a solicitação de retirada de um bem patrimonial para uso em espaço externo à unidade. Para registro de empréstimos, o usuário deve necessariamente informar a pessoa responsável, podendo tratar-se de um servidor ou de um estudante da FACOM. Este é o momento, no fluxo do sistema, em que são gerados termos de responsabilidade pela posse do item em questão (Figuras 4.6 e 4.7), em atendimento a um dos principais requisitos funcionais inicialmente levantados.

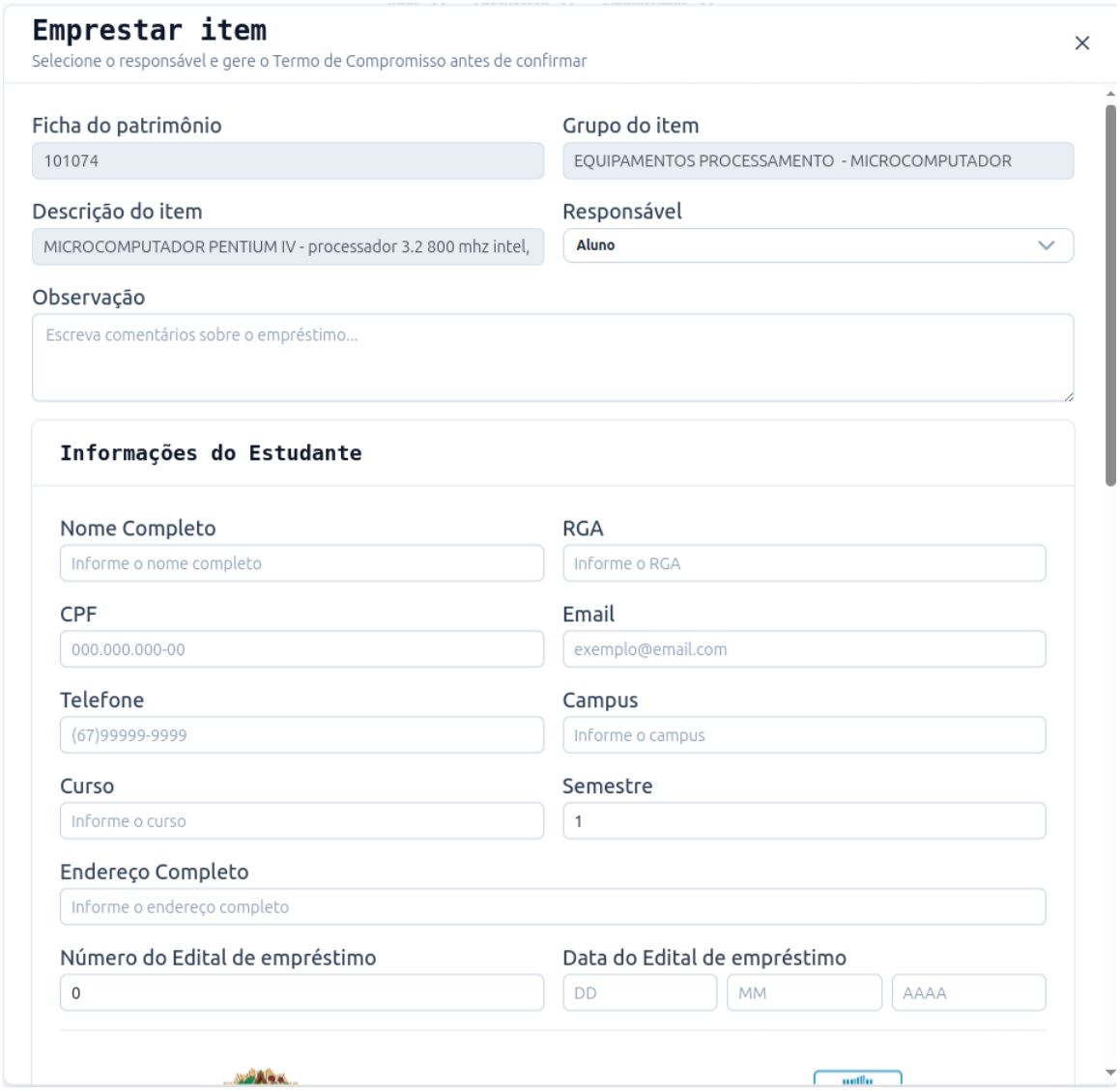

Figura 4.6: Formulário para registro de empréstimo.

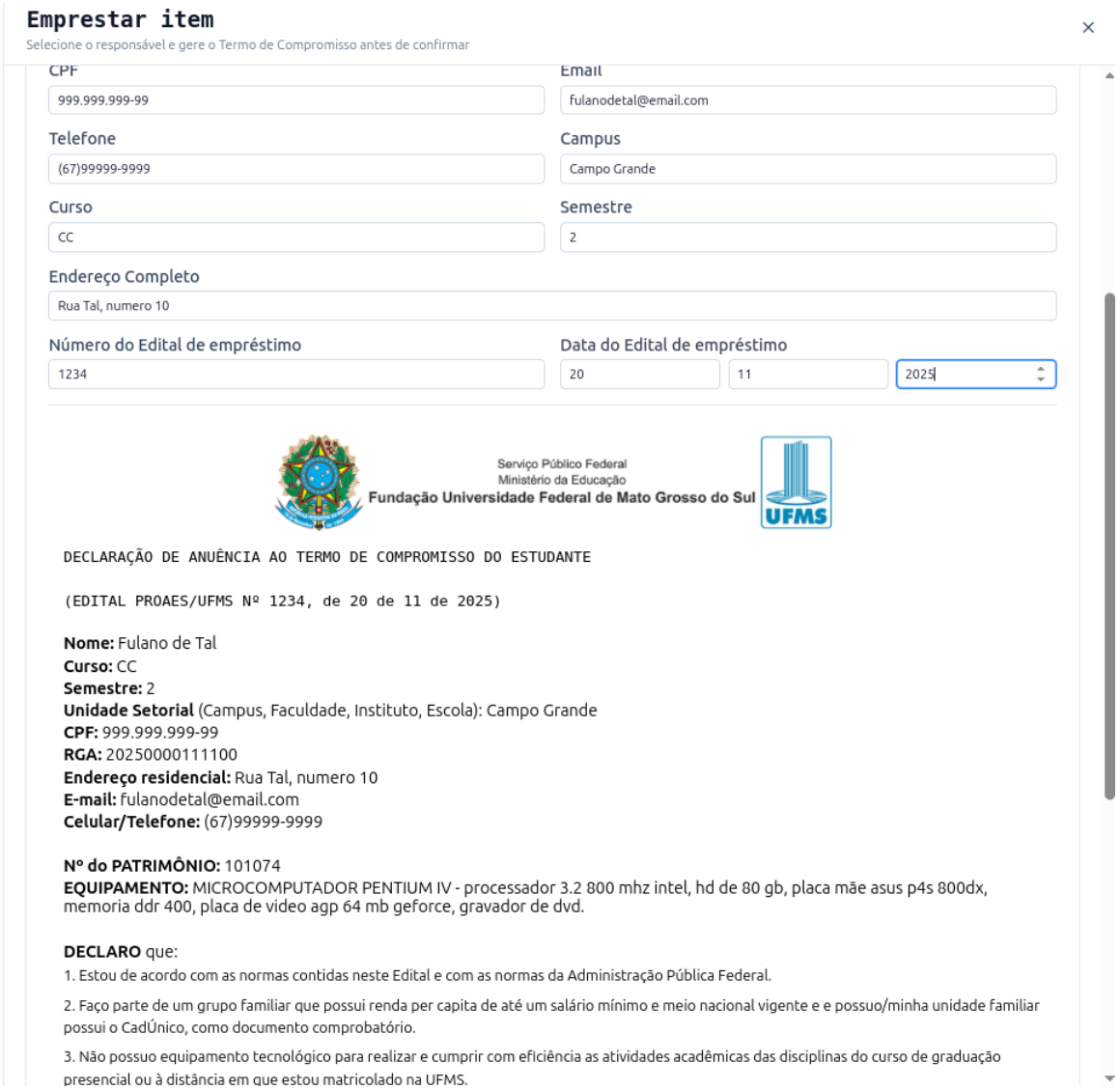

Figura 4.7: Geração automática do Termo de Empréstimo.

4.4 Módulo de gerenciamento dos Locais, Salas e Responsáveis

Estes módulos operam de maneira complementar aos anteriores, porém, não menos importantes, carregam consigo os dados necessários para compor as informações das movimentações.

Foram definidos quatro locais para subdividir o espaço macro de gerência dos bens patrimoniais, a saber: "FACOM - Térreo", "FACOM - Andar 1", "FACOM - Andar 2"e "FACOM - Externo". No contexto do projeto, os Locais são espaços agrupadores de Salas, onde cada Sala deve ter um servidor responsável associado, elegido internamente como corresponsável pelo controle e gestão dos itens alocados naquela Sala.

Diante disso, conforme demandado como requisito funcional, o sistema implementa a geração de termos de corresponsabilidade, em que é atribuída ao servidor elegido a competência de guardar e zelar pelos bens patrimoniais alocados em cada sala (Figuras 4.8 e 4.9).

Figura 4.8: Imagem ilustrativa da exibição de Salas.

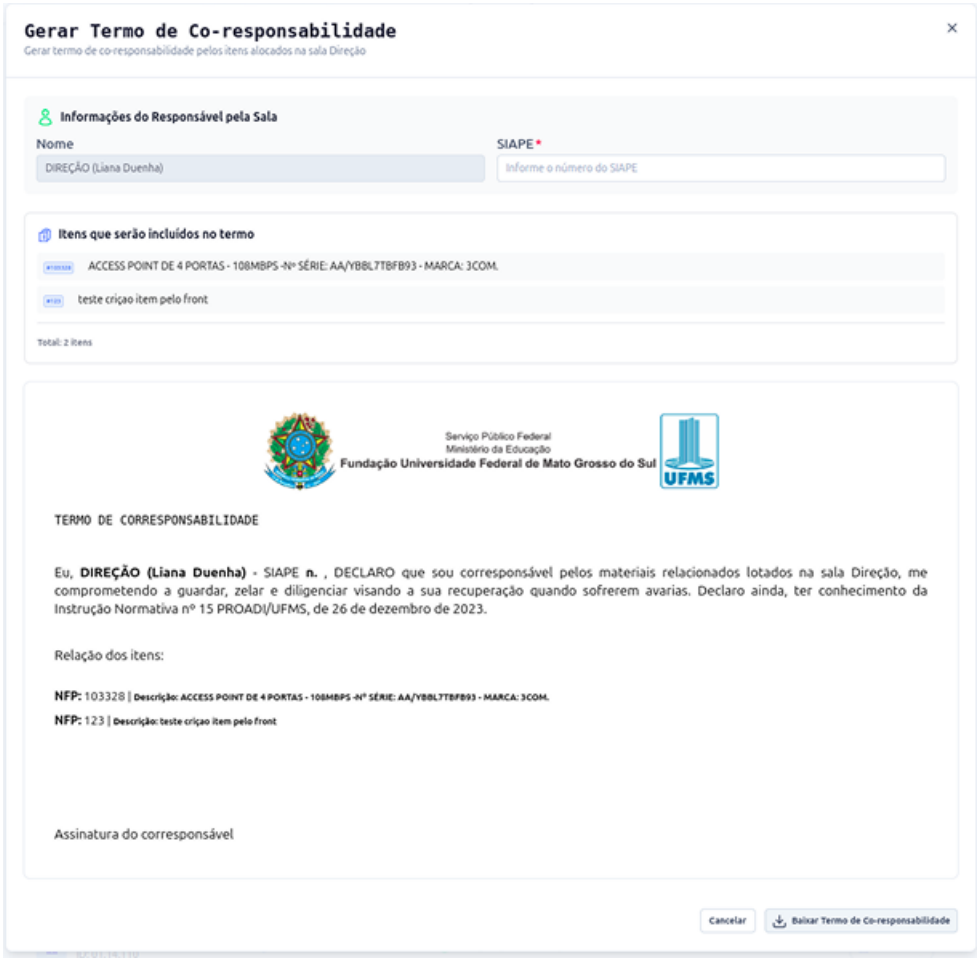

Figura 4.9: Geração automática do Termo de Corresponsabilidade pelos itens da Sala.

Capítulo 5

Considerações Finais

A execução do projeto objeto deste trabalho alcançou o objetivo geral proposto: a construção de um sistema *web* moderno que possibilita a gestão e o controle interno do patrimônio material da FACOM, a partir de uma base de dados coesa com os dados oficiais da UFMS, com a importante capacidade de demonstrar a exata localização física de cada item.

Os objetivos específicos também foram atendidos, com destaque para os módulos de gerenciamento do ciclo de vida dos itens e de suas possíveis movimentações; bem como para a possibilidade de geração de documentos que atribuem a pessoas uma corresponsabilidade por itens em seu poder de uso, além da geração de relatórios com dados completos dos itens.

A solução implementada demonstra contribuição direta para a modernização da gestão patrimonial da FACOM/UFMS, centralizando os dados oficiais dos bens e gerenciando suas possíveis movimentações, facilitando o controle de sua localização e eliminando disparidades entre os dados do sistema interno e os da UFMS.

Verificou-se que a execução do projeto implementou aplicações práticas de conceitos e metodologias da Engenharia de *Software* no desenvolvimento de um sistema *web* para solucionar um problema real da instituição. A aplicação da arquitetura em camadas e a utilização de tecnologias modernas, que favorecem a experiência do usuário, revelam uma estrutura sólida para a manutenção, a escalabilidade, a implantação e a evolução do *software*.

Assim, o pacote entregue atende ao conjunto de requisitos delimitados para a versão 1.0, fornecendo um sistema funcional e pronto para a implantação.

5.1 Limitações e Evolução incremental

Algumas dificuldades foram identificadas ao longo do desenvolvimento do projeto, a saber:

- Não foi possível automatizar a integração do sistema diretamente com a base de dados patrimonial da UFMS, porque, por motivos de segurança, a Universidade não disponibiliza APIs para expor tais informações. Assim, foi necessário desenvolver uma solução própria para processar esses dados a partir dos relatórios de inventário anual (2025) da Universidade, em formato CSV.

- A ausência de ambiente de homologação para a execução de testes pelos usuários finais impossibilitou a validação formal das funcionalidades, que ficou restrita a tes-

tes automatizados no *backend* e exploratórios no *frontend*, executados em ambiente local.

- Além disso, em virtude do curto período de desenvolvimento do trabalho, algumas funcionalidades mapeadas inicialmente precisaram ser despriorizadas para a versão 1.0 de entrega. Por exemplo, o módulo de gerenciamento de usuários e permissões e de autenticação.

5.2 Trabalhos Futuros

Diante das limitações citadas e da necessidade de melhorias contínuas da gestão da FACOM, abrem-se portas para a evolução e aprimoramento do sistema em trabalhos futuros, como:

- Implementação de uma API dedicada à integração direta com a base de dados patrimonial da UFMS;
- Implementação do módulo de gerenciamento de usuários/permissões e autenticação;
- Finalização dos módulos de gerenciamento dos Locais, Salas e Responsáveis;
- Melhorias contínuas de usabilidade com base na avaliação da experiência do usuário.

Capítulo 6

Referências Bibliográficas

[1] B. Frost. *Atomic design*. Brad Frost, 2016.

[2] Denise Carvalho and Marcos Ceccato. *Manual completo de contabilidade pública*. Elsevier, Rio de Janeiro, 2011.

[3] I. Sommerville. *Software Engineering*. International computer science series. Pearson, 2011.

[4] R.S. Pressman and B.R. Maxim. *Engenharia de software - 9.ed.* McGraw Hill Brasil, 2021.

[5] S.R. Palmer and J.M. Felsing. *A Practical Guide to Feature-driven Development*. Coad series. Prentice Hall PTR, 2002.

[6] M. Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley signature series. Addison-Wesley, 2010.

[7] S. Chacon and B. Straub. *Pro Git*. Apress, 2014.

[8] Laura Dabbish, Les Gasser, and James Herbsleb. The social side of software engineering. *IEEE Software*, 34(3):40–47, 2017.

[9] Mohamed Fayad and Douglas Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, 1997.

[10] Nuxt. Nuxt Documentation. Disponível em: `https://nuxt.com/docs/getting-started/introduction`, 2024. Acesso em: 1 nov. 2025.

[11] VueJS. VueJS Documentation. Disponível em: `https://vuejs.org/guide/introduction.html`, 2024. Acesso em: 1 nov. 2025.

[12] M. Fowler. *Patterns of Enterprise Application Architecture*. A Martin Fowler signature book. Addison-Wesley, 2003.

[13] A. Osmani. *Learning JavaScript Design Patterns*. JavaScript and jQuery developer's guide. O'Reilly Media, Incorporated, 2012.

[14] Nuxt UI. Nuxt UI Documentation. Disponível em: `https://ui.nuxt.com/`, 2023. Acesso em: 1 nov. 2025.

[15] Tailwind Labs. Tailwind CSS Documentation. Disponível em: https://tailwindcss.com/docs, 2024. Acesso em: 1 nov. 2025.

[16] Prisma. Prisma Documentation. Disponível em: https://www.prisma.io/docs/, 2024. Acesso em: 1 nov. 2025.

[17] Prettier. Prettier Documentation. Disponível em: https://prettier.io/docs/en/, 2025. Acesso em: 1 nov. 2025.

[18] ESLint. ESLint Documentation. Disponível em: https://eslint.org/docs/latest/, 2025. Acesso em: 1 nov. 2025.

[19] Amanda Vieira. Projeto do sistema para gestão patrimonial da facom/ufms. [Software]. Versão 1.0. Disponível em: https://github.com/eldermendes/patrimonio-facom-ufms, 2025. Último commit: 8 dez. 2025. Acesso em: 8 dez. 2025.