



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CIÊNCIAS DA COMPUTAÇÃO

Gustavo Gonçalves dos Santos

**Sistema Web para Gerenciamento Compartilhado de um Laboratório de Pesquisa e
Desenvolvimento**

Florianópolis
2026

Gustavo Gonçalves dos Santos

Sistema Web para Gerenciamento Compartilhado de um Laboratório de Pesquisa e Desenvolvimento

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciências da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Ismael Seidel, Dr.

Florianópolis

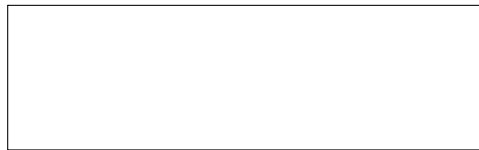
2026

Gustavo Gonçalves dos Santos

**Sistema Web para Gerenciamento Compartilhado de um Laboratório de Pesquisa e
Desenvolvimento**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo curso de Graduação em Ciências da Computação.

Florianópolis, 11 de julho de 2026.



Coordenação do Curso

Banca Examinadora:



Prof. Ismael Seidel, Dr.
Orientador



Profa. Cristina Meinhardt, Dra.
Instituição Universidade Federal de Santa Catarina



Prof. José Luís Almada Güntzel, Dr.
Instituição Universidade Federal de Santa Catarina

Florianópolis, 2026.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer à minha mãe, Cristiane, que sempre esteve presente para ouvir minhas preocupações e dificuldades ao longo da graduação. Agradeço também ao meu pai, Lucio, pela compreensão e apoio nos momentos mais desafiadores dessa trajetória. Estendo meus agradecimentos aos amigos e colegas que fizeram parte desta etapa da minha vida e que estiveram ao meu lado quando mais precisei. Em especial, agradeço ao meu amigo de infância, Thiago, que sempre me auxiliou na tomada de decisões importantes, tanto no âmbito acadêmico quanto profissional. Agradeço ao meu orientador, Ismael, por encontrar um tema de TCC na área de desenvolvimento web e por me auxiliar durante esse período final da graduação. Gostaria também de agradecer a todos os integrantes da minha antiga equipe do LabSec, pela compreensão e apoio durante minha formação acadêmica. Por fim, agradeço a todos da Onsign, especialmente ao meu gerente, Eduardo, que sempre demonstraram compreensão em relação aos meus horários diferenciados e contribuíram significativamente para meu crescimento profissional e desenvolvimento como desenvolvedor.

“There is still much to come, and each step taken so far is part of a larger journey.”

RESUMO

O gerenciamento eficiente de laboratórios de pesquisa e desenvolvimento universitários apresenta desafios relacionados à organização e controle de recursos, bem como à administração de equipes. Atualmente, muitos laboratórios utilizam métodos manuais ou ferramentas não integradas, o que pode levar à falta de controle adequado sobre equipamentos, materiais e membros da equipe. O estado da arte revela que há uma carência de sistemas que centralizem todas as atividades necessárias para uma gestão colaborativa e eficiente de laboratórios. Este trabalho apresenta o desenvolvimento de um sistema web para o gerenciamento compartilhado de laboratórios de pesquisa e desenvolvimento, oferecendo funcionalidades como o controle de integrantes (graduandos, mestrandos, doutorandos) e a gestão de materiais, todos identificados por etiquetas de código. A solução foi construída com um *stack* tecnológico moderno e amplamente utilizado na área de desenvolvimento web, empregando Django e Python no *backend*, React e TypeScript no *frontend* e PostgreSQL como banco de dados, fornecendo uma plataforma centralizada e de fácil acesso que aumenta a eficiência e a transparência nas operações do laboratório. Os resultados obtidos incluem a melhoria no controle dos recursos e a redução do tempo gasto com tarefas administrativas.

Palavras-chave: Gerenciamento de Laboratórios, Sistema Web, Gestão de Recursos.

ABSTRACT

The efficient management of university research and development laboratories presents challenges related to the organization and control of resources, as well as the administration of teams. Currently, many laboratories use manual methods or non-integrated tools, which can lead to a lack of adequate control over equipment, materials and team members. The state of the art reveals that there is a lack of systems that centralize all the activities necessary for collaborative and efficient laboratory management. This work presents the development of a web system for the shared management of research and development laboratories, offering functionalities such as control of members (undergraduate, master's and doctoral students) and management of materials, all identified by code labels. The solution was built with a modern technology stack widely used in web development, employing Django and Python for the backend, React and TypeScript for the frontend, and PostgreSQL as the database, providing a centralized and easily accessible platform that increases the efficiency and transparency of laboratory operations. The results obtained include improved resource control and a reduction in the time spent on administrative tasks.

Keywords: Laboratory Management, Web System, Resource Management.

LISTA DE FIGURAS

Figura 1 – Comparação entre infraestrutura local e em nuvem para hospedagem de aplicações web.	16
Figura 2 – Arquitetura simplificada de um sistema web com modelo cliente-servidor. .	17
Figura 3 – Fluxo simplificado de autenticação de usuários no sistema web.	19
Figura 4 – Fluxo geral de interação entre as camadas do sistema.	24
Figura 5 – Arquitetura geral do sistema.	27
Figura 6 – Interface de gerenciamento de membros do laboratório.	32
Figura 7 – Interface de gestão de equipamentos do laboratório.	33
Figura 8 – Interface de geração de relatórios.	34
Figura 9 – Página pública do laboratório.	35
Figura 10 – Página pública do laboratório acessada em dispositivo móvel.	36

LISTA DE TABELAS

Tabela 1 – Comparativo entre os trabalhos relacionados e a solução proposta.	21
--	----

LISTA DE ABREVIATURAS E SIGLAS

2FA	Autenticação em Dois Fatores – <i>Two Factor Authentication</i>	18
API	<i>Application Programming Interface</i>	13
AWS	<i>Amazon Web Services</i>	14
CSRF	<i>Cross-Site Request Forgery</i>	18
DOM	<i>Document Object Model</i>	13
HTML	Linguagem de Marcação de Hipertexto – <i>HyperText Markup Language</i>	17
JSON	Notação de Objeto JavaScript – <i>JavaScript Object Notation</i>	17
JWT	Token Web JSON – <i>JSON Web Token</i>	28
LAN	Rede de Área Local – <i>Local Area Network</i>	17
MVT	<i>Model-View-Template</i>	13
OCI	<i>Oracle Cloud Infrastructure</i>	14
SPA	Aplicação de Página Única – <i>Single Page Application</i>	26
SSO	<i>Single Sign-On</i>	18
WBMLS	<i>Web-based Laboratory Management System</i>	19

SUMÁRIO

1	INTRODUÇÃO	12
1.1	OBJETIVOS	12
1.1.1	Objetivo geral	12
1.1.2	Objetivos específicos	13
1.2	MÉTODO	13
1.3	ORGANIZAÇÃO DO TRABALHO	14
2	CONCEITOS FUNDAMENTAIS E TRABALHOS RELACIONADOS .	15
2.1	SISTEMAS WEB	15
2.1.1	Modelo cliente-servidor	16
2.1.2	Autenticação	18
2.2	TRABALHOS RELACIONADOS	19
2.2.1	<i>Web-based Laboratory Management System - Federal Polytechnic, Níger .</i>	<i>19</i>
2.2.2	<i>Cadets Administration System - Nusantara Maritime Academy</i>	<i>20</i>
2.2.3	Avaliação qualitativa em laboratório clínico	20
2.2.4	Comparativo dos trabalhos relacionados	20
3	REQUISITOS DA APLICAÇÃO E MODELAGEM	22
3.1	REQUISITOS DA APLICAÇÃO	22
3.1.1	Requisitos Funcionais	22
3.1.2	Requisitos Não Funcionais	23
3.2	MODELAGEM	24
4	DESENVOLVIMENTO	26
4.1	ARQUITETURA DO SISTEMA	26
4.2	BACKEND	27
4.2.1	Módulo de contas (accounts)	27
4.2.2	Módulo de autenticação (oauth)	28
4.2.3	Módulo de conteúdo (content)	28
4.2.4	Módulo de configurações do laboratório (core)	29
4.2.5	Utilitários transversais (sitewide)	29
4.3	FRONTEND	30
4.3.1	Estrutura de rotas e controle de acesso	30
4.3.2	Gerenciamento de estado global	30
4.3.3	Sistema de modais	31
4.3.4	Internacionalização	31
4.4	FUNCIONALIDADES IMPLEMENTADAS	31

4.5	DISTRIBUIÇÃO E IMPLANTAÇÃO	37
5	CONCLUSÕES	38
	Referências	40
	APÊNDICES	42
	APÊNDICE A – ARTIGO	43
	APÊNDICE B – INSTRUÇÕES PARA EXECUÇÃO LOCAL DO PROJETO	54
B.1	CONFIGURAÇÃO DO AMBIENTE PYTHON	54
B.2	CONFIGURAÇÃO DO FRONTEND	54
B.3	CONFIGURAÇÃO DO BANCO DE DADOS	54
B.4	INICIALIZAÇÃO DO BACKEND	55
B.5	EXECUÇÃO DO SERVIDOR	55
	APÊNDICE C – INSTRUÇÕES PARA IMPLANTAÇÃO EM AMBIENTE DE PRODUÇÃO	56
C.1	PRÉ-REQUISITOS DO SERVIDOR	56
C.2	INSTALAÇÃO DE DEPENDÊNCIAS DO SISTEMA	56
C.3	CONFIGURAÇÃO DO POSTGRESQL	57
C.4	OBTENÇÃO DO CÓDIGO-FONTE	57
C.5	CONFIGURAÇÃO DO AMBIENTE PYTHON	57
C.6	BUILD DO FRONTEND	57
C.7	ARQUIVO DE CONFIGURAÇÃO DE AMBIENTE	58
C.8	VERIFICAÇÃO DAS CONFIGURAÇÕES DE PRODUÇÃO	58
C.9	PREPARAÇÃO DO DJANGO	59
C.10	CONFIGURAÇÃO DO UWSGI PARA PRODUÇÃO	59
C.11	CONFIGURAÇÃO DO NGINX	60
C.12	SERVIÇO SYSTEMD PARA O UWSGI	61
C.13	CERTIFICADO SSL COM LET'S ENCRYPT	61
C.14	CONFIGURAÇÃO DO FIREWALL	62
C.15	PERMISSÕES DE ARQUIVOS	62

1 INTRODUÇÃO

O gerenciamento eficiente de laboratórios de pesquisa e desenvolvimento é cada vez mais relevante e necessário no contexto universitário. Afinal de contas, além das tarefas de pesquisa, tais laboratórios frequentemente enfrentam desafios relacionados à organização e controle de materiais e equipamentos (Akeem Adekunle et al., 2024; Timoteo et al., 2020). Além disso, é necessária administração de equipes compostas por alunos de diferentes níveis de formação, como graduação, mestrado e doutorado, e até mesmo técnicos e professores.

Métodos tradicionais, de planilhas digitais até registros manuais através de papel e caneta, muitas vezes podem resultar em processos pouco eficientes e em uma falta de controle adequado sobre os recursos e materiais disponíveis (Fadilah et al., 2021; Akeem Adekunle et al., 2024). Por conta disso, pode existir um comprometimento tanto na produtividade quanto na transparência da utilização dos recursos do laboratório.

Nesse contexto, soluções digitais baseadas na web vêm ganhando destaque como uma alternativa promissora. Aplicações web oferecem, entre outras vantagens, acessibilidade remota, interfaces mais intuitivas e a possibilidade de reunir múltiplas funcionalidades em um só lugar (Rosado; Metrolho; Ribeiro, 2016). Essas características, quando bem exploradas, tendem a tornar esse tipo de sistema uma opção viável para lidar com as demandas específicas de um laboratório acadêmico.

Além disso, tecnologias modernas costumam oferecer um bom nível de flexibilidade e escalabilidade, o que pode facilitar futuras adaptações e expansões do sistema, à medida que surgem novas necessidades. Dessa forma, não se trata apenas de resolver problemas imediatos de organização, mas também de criar uma base tecnológica que favoreça melhorias contínuas nos processos de gestão e colaboração dentro dos laboratórios.

1.1 OBJETIVOS

1.1.1 Objetivo geral

O objetivo geral deste trabalho foi desenvolver um sistema web para o gerenciamento compartilhado de laboratórios de pesquisa e desenvolvimento. O sistema promove uma gestão eficiente, colaborativa e centralizada de integrantes, materiais e recursos, com foco na melhoria da organização, controle e transparência das operações laboratoriais. Além disso, o sistema serve também como fonte de dados para a apresentação do laboratório, através de um site público.

1.1.2 Objetivos específicos

- O1** Projetar uma plataforma web que permita o cadastro, edição e visualização de informações sobre os integrantes do laboratório, como graduandos, mestrandos e doutorandos.
- O2** Implementar e validar um sistema de gestão de materiais com etiquetas de código para identificar e rastrear equipamentos e recursos do laboratório.
- O3** Criar uma interface de usuário responsiva e intuitiva, para facilitar o acesso e o uso do sistema pelos integrantes do laboratório.
- O4** Desenvolver *Application Programming Interfaces* (APIs) robustas e bem documentadas que permitam a utilização a partir de variadas implementações de *frontend*, garantindo a segurança, a integridade dos dados e a escalabilidade do sistema.
- O5** Incorporar funcionalidades adicionais conforme as demandas específicas surgirem durante o desenvolvimento, como gerenciamento de integrantes em projetos, convites para novos integrantes ou geração de relatórios.
- O6** Garantir a integração e o funcionamento colaborativo do sistema em diferentes dispositivos e plataformas, promovendo acessibilidade e mobilidade.

1.2 MÉTODO

A construção do sistema adotou uma abordagem flexível, já que diferentes demandas poderiam surgir no decorrer do desenvolvimento. Portanto, a inserção de novas funcionalidades e módulos foi possível ao longo do projeto, com testes customizados para ambos o *backend* e o *frontend*.

A parte do *backend* do sistema foi estruturada com o *framework* Django (Django, 2025), que combina segurança, robustez e uma arquitetura clara baseada no padrão *Model-View-Template* (MVT). Por ser construído sobre Python, que é uma linguagem cada vez mais presente no cenário acadêmico e corporativo (Vincent, 2022), esse *framework* facilitou não só o desenvolvimento técnico, mas também a compreensão e manutenção do código por parte de novos colaboradores. Seu sistema de administração integrado e o suporte nativo à criação de APIs foram elementos que contribuíram para tornar o processo mais fluido.

Para a parte do *frontend* foi utilizada a biblioteca React (React, 2024), cuja abordagem baseada em componentes e virtual *Document Object Model* (DOM) favoreceu o desenvolvimento de interfaces dinâmicas e interativas. Além disso, para integrar o servidor do *backend* com a interface React, foi utilizado Vite (VoidZero, 2024) e APIs de requisições, garantindo o funcionamento seguro e eficiente de ambas as partes do sistema.

Para armazenar e gerenciar os dados da aplicação, foi utilizado o banco de dados relacional PostgreSQL (PostgreSQL, 2024). Trata-se de uma tecnologia que, além de consolidada,

oferece recursos avançados de integridade transacional e suporte a tipos de dados complexos, mostrando-se adequada diante da diversidade de informações gerenciadas pelo sistema, como registros de usuários, materiais e equipamentos cadastrados, configurações específicas do laboratório, etc.

Quanto à infraestrutura de hospedagem, foi elaborado um guia para implantação do sistema em um servidor Linux com uWSGI e Nginx, disponível no Apêndice C. Essa abordagem é igualmente aplicável em plataformas em nuvem, como *Amazon Web Services* (AWS) ou *Oracle Cloud Infrastructure* (OCI), que oferecem escalabilidade sob demanda, alta disponibilidade e acesso remoto contínuo (Chieu et al., 2009).

Ao longo do desenvolvimento, práticas de controle de versão foram adotadas com o uso de Git, mantendo-se o código-fonte em um repositório remoto no GitHub. Essa organização facilitou a rastreabilidade de mudanças e correções ao decorrer do desenvolvimento.

Por fim, testes foram conduzidos de forma contínua ao longo do desenvolvimento, com atenção à parte funcional do sistema. Testes unitários e de integração foram realizados para identificar falhas e garantir a estabilidade do sistema. Uma avaliação prática com usuários finais do laboratório, como bolsistas e pesquisadores, foi identificada como trabalho futuro, conforme discutido no Capítulo 5.

1.3 ORGANIZAÇÃO DO TRABALHO

O restante deste trabalho está organizado como segue. O Capítulo 2 apresenta os conceitos fundamentais que embasam o desenvolvimento, abordando sistemas web, o modelo cliente-servidor e os mecanismos de autenticação, além de discutir trabalhos relacionados e um comparativo entre eles. O Capítulo 3 descreve os requisitos funcionais e não funcionais identificados para a aplicação e apresenta a modelagem do sistema, detalhando a organização das camadas de *frontend*, *backend* e banco de dados. O Capítulo 4 detalha o processo de desenvolvimento, incluindo a arquitetura adotada, a implementação do *backend* em Django e do *frontend* em React, as funcionalidades entregues e os procedimentos de distribuição e implantação. Finalmente, o Capítulo 5 apresenta as conclusões obtidas e aponta direções para trabalhos futuros.

2 CONCEITOS FUNDAMENTAIS E TRABALHOS RELACIONADOS

Este capítulo inicia com a apresentação dos conceitos fundamentais deste trabalho e conclui com uma breve descrição de trabalhos relacionados.

2.1 SISTEMAS WEB

A noção de sistemas web pode ser entendida sob múltiplas perspectivas, variando conforme o foco, seja técnico, organizacional ou de uso. De maneira geral, trata-se de um tipo de aplicação que opera em ambiente de rede e é acessada através de navegadores, muitas vezes estruturada sobre arquiteturas distribuídas e que se apoia em múltiplas camadas. Segundo Garousi et al. (2013), o desenvolvimento de aplicações web, assim como o de sistemas tradicionais, exige processos complexos que envolvem coleta de requisitos, múltiplas linguagens e tecnologias, o que leva a uma considerável heterogeneidade técnica e arquitetural.

Nesse sentido, é possível caracterizar uma aplicação web como um sistema composto por duas partes principais, uma interface acessível via navegador (*frontend*), onde os usuários realizam suas interações, e um servidor de aplicação (*backend*), responsável por processar as requisições, aplicar regras de negócio, realizar autenticações e acessar os dados armazenados na base de dados. Essas aplicações podem se apresentar sob formas estáticas ou dinâmicas, variando conforme o tipo de resposta que o sistema deve entregar frente às ações do usuário. Nos últimos anos, têm se consolidado como ferramentas fundamentais em setores como comércio eletrônico, bancos e serviços acadêmicos, justamente pela capacidade de oferecer acesso remoto, interação em tempo real e automatização de processos (Molina-Ríos; Pedreira-Souto, 2020).

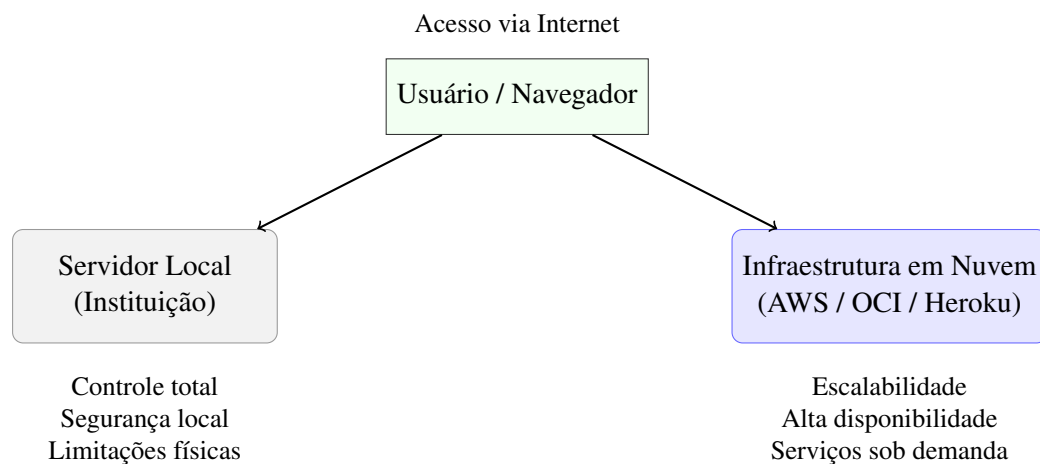
Al-Salem; Abu Samaha (2007) destacam algumas das características mais marcantes dos sistemas web, como os requisitos frequentemente voláteis¹, a presença de múltiplos perfis de usuários (muitos deles anônimos), e a complexidade de conteúdo e estrutura. Além disso, apontam que o ciclo de desenvolvimento costuma ser acelerado, geralmente inferior a três meses, embora cada etapa envolva desafios específicos que exigem atenção criteriosa por parte dos desenvolvedores.

Em contextos institucionais, como o de universidades, essas características fazem com que os sistemas web sejam particularmente úteis para lidar com fluxos informacionais diversos, demandas de acesso remoto e a necessidade de colaboração entre usuários com diferentes níveis de envolvimento. A combinação entre modularidade, escalabilidade e acessibilidade reforça a posição dessas aplicações como solução viável (ou até preferencial) para problemas de gerenciamento, organização e comunicação em ambientes digitais (Rosado; Metrolho; Ribeiro, 2016).

¹Requisitos voláteis são aqueles que tendem a mudar com frequência ao longo do desenvolvimento do sistema, geralmente por causa de novas demandas dos usuários, mudanças no escopo do projeto ou atualizações tecnológicas.

Outro ponto que merece atenção é a dependência da infraestrutura em nuvem. Embora a computação em nuvem tenha se consolidado como o principal meio de hospedagem e escalonamento de aplicações web (Chieu et al., 2009), ainda há instituições que preferem (ou precisam) manter parte de suas soluções em servidores locais, por questões de segurança, política interna ou limitações orçamentárias. Mesmo assim, serviços como AWS, Oracle OCI e Heroku têm sido amplamente utilizados por universidades, *startups* e centros de pesquisa devido à sua capacidade de oferecer disponibilidade contínua, elasticidade e automação. A Figura 1 apresenta um diagrama comparativo, tal que ambas as abordagens podem ser consideradas neste trabalho.

Figura 1 – Comparação entre infraestrutura local e em nuvem para hospedagem de aplicações web.



Fonte: o autor.

No contexto da administração de laboratórios acadêmicos, os sistemas web podem desempenhar um papel relevante ao centralizar informações, facilitar o compartilhamento de dados e oferecer interfaces acessíveis a partir de diferentes dispositivos. A possibilidade de acessar o sistema de forma remota, seja de casa, do campus ou de um dispositivo móvel, acaba por contribuir para uma gestão mais fluida e colaborativa, especialmente quando há múltiplos usuários gerenciando e atuando sobre os mesmos recursos.

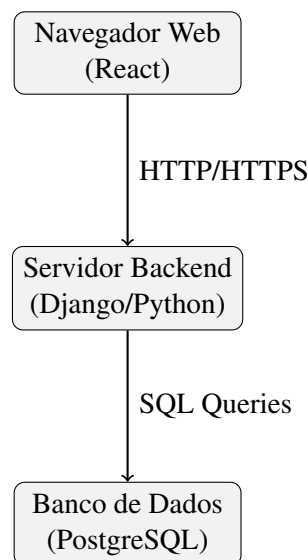
2.1.1 Modelo cliente-servidor

Nem sempre é possível visualizar, à primeira vista, o que acontece por trás de uma simples interação em uma página web. No entanto, por trás do clique, da requisição, da resposta que chega quase instantaneamente, existe uma estrutura de comunicação que há décadas serve de base para boa parte das aplicações distribuídas que usamos e isso pode ser chamado de modelo cliente-servidor.

Esse modelo, longe de ser uma ideia recente, continua extremamente relevante, mesmo com o surgimento de arquiteturas mais descentralizadas, como *peer-to-peer* ou *serverless*. Ele parte de um princípio bastante direto, mas ao mesmo tempo poderoso: uma entidade (o cliente)

faz uma solicitação, enquanto outra (o servidor) responde. A Figura 2 apresenta um exemplo da interação entre cliente e servidor. Mas esse equilíbrio de papéis, embora pareça simples, carrega consigo implicações concretas em diversas dimensões: em termos de escalabilidade, à medida que o número de clientes simultâneos cresce, o servidor pode se tornar um gargalo, exigindo estratégias como balanceamento de carga ou replicação horizontal; em segurança, toda a comunicação precisa ser protegida por protocolos como HTTPS, com controle rigoroso de sessões e permissões de acesso; e em controle de fluxo, mecanismos como *cache* e limitação de requisições tornam-se essenciais para manter a estabilidade e a responsividade do serviço (Wu; Lu; Chen, 2024).

Figura 2 – Arquitetura simplificada de um sistema web com modelo cliente-servidor.



Fonte: o autor.

Na prática, o cliente pode assumir diversas formas, pode ser um navegador acessando um sistema web, um aplicativo *mobile* requisitando dados, ou mesmo outro servidor intermediário. Do outro lado, o servidor é o responsável por processar as requisições, acessar recursos (como bancos de dados, arquivos ou serviços externos) e devolver ao cliente aquilo que ele pediu, seja uma página em Linguagem de Marcação de Hipertexto – *HyperText Markup Language* (HTML), um conjunto de dados em Notação de Objeto JavaScript – *JavaScript Object Notation* (JSON) ou até uma imagem processada em tempo real.

Evidentemente, essa relação nem sempre é estática ou rígida. Dependendo da arquitetura adotada, um mesmo agente pode funcionar ora como cliente, ora como servidor. É o que ocorre, por exemplo, em micro-serviços distribuídos ou em requisições para APIs que se consomem mutuamente. Ainda assim, o modelo cliente-servidor permanece como uma abstração útil para organizar responsabilidades e entender o fluxo de dados em sistemas conectados.

Do ponto de vista histórico, a consolidação desse modelo ocorreu principalmente a partir da década de 1980, com o avanço da Rede de Área Local – *Local Area Network* (LAN) e o surgimento dos primeiros sistemas de arquivos distribuídos. A partir daí, com o crescimento

da internet, a arquitetura cliente-servidor passou a sustentar grande parte da comunicação entre dispositivos remotos. Autores como Tanenbaum; Van Steen (2007) apontam que esse modelo, embora sujeito a gargalos do lado servidor, ainda representa uma estrutura sólida para aplicações que exigem consistência e centralização.

Outro fator relevante e que ajuda a explicar a persistência do modelo é sua compatibilidade com os paradigmas modernos de desenvolvimento web. *Frameworks* como Django, por exemplo, seguem esse padrão ao separar as requisições HTTP (clientes) do processamento e renderização realizados pelo *backend* (servidor). O mesmo se aplica a bibliotecas *frontend* como React, que consomem rotas e dados providos por serviços web hospedados em servidores.

Por outro lado, alguns especialistas destacam que o modelo cliente-servidor pode não se ajustar bem a sistemas que exigem alta disponibilidade descentralizada, ou àqueles que dependem fortemente de comunicação em tempo real bidirecional (Wu; Lu; Chen, 2024). Mesmo assim, sua simplicidade conceitual e clareza de funções tornam-no ainda hoje uma das bases mais sólidas e utilizadas em aplicações web contemporâneas.

2.1.2 Autenticação

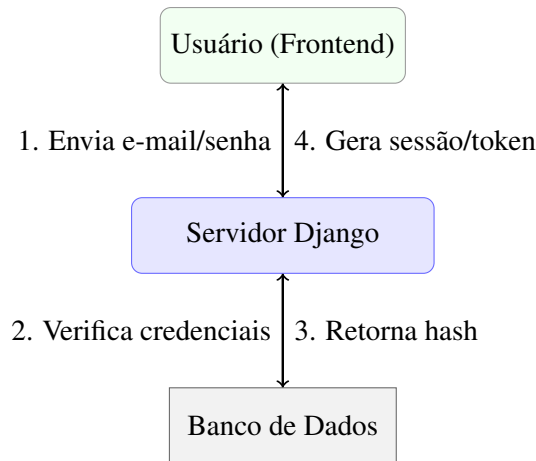
Dentre as várias funcionalidades que compõem um sistema web, poucas são tão críticas quanto os mecanismos de autenticação (OWASP Foundation, 2021). Em um contexto onde informações sensíveis, permissões específicas e fluxos colaborativos precisam ser controlados de forma segura, estabelecer uma maneira confiável de verificar a identidade dos usuários é mais do que uma questão técnica, é um requisito de segurança e organização.

De modo geral, a autenticação consiste no processo de identificar e validar um usuário com base em algum tipo de credencial, geralmente combinando identificador (como *e-mail* ou nome de usuário) e senha. No entanto, com a constante sofisticação de ataques e a diversificação de contextos de acesso, esse processo tem evoluído para incorporar camadas adicionais, como Autenticação em Dois Fatores – *Two Factor Authentication* (2FA), *tokens* temporários, e autenticação via redes sociais ou instituições externas com o uso de *Single Sign-On* (SSO).

No sistema implementado, ilustrado na Figura 3, a autenticação dos usuários segue um fluxo tradicional com uso de *e-mail* e senha, armazenados de forma segura no banco de dados com o uso de *tokens* *Cross-Site Request Forgery* (CSRF). A camada de autenticação foi mantida suficientemente flexível, de modo a possibilitar a adoção futura de outros métodos, como autenticação via provedores externos ou até mesmo a integração com sistemas de autenticação centralizada já utilizados, por exemplo pela UFSC (UFSC, 2024).

A autenticação ocorre no *backend*, com Django lidando com a verificação das credenciais, emissão de *tokens* de sessão e controle de permissões (Django, 2025). O *frontend*, por sua vez, é responsável por armazenar temporariamente os dados de sessão (como *cookies* ou *tokens* de autenticação) e redirecionar os usuários para áreas permitidas conforme seu status.

Figura 3 – Fluxo simplificado de autenticação de usuários no sistema web.



Fonte: o autor.

Também deve ser destacado que a escolha de um método de autenticação não deve se basear apenas na conveniência técnica, mas também na natureza dos dados envolvidos. Em ambientes acadêmicos, onde o acesso pode incluir informações de projetos, equipamentos compartilhados ou documentos de pesquisa, falhas nesse processo podem comprometer não só o sistema, mas também o funcionamento coletivo do laboratório.

Além disso, um sistema de autenticação bem desenhado também pode servir como base para funções mais avançadas, como controle de permissões por papéis, auditoria de ações realizadas no sistema ou integração com ferramentas externas. Ao pensar a autenticação como um ponto de entrada estratégico e não apenas um formulário de *login*, amplia-se o potencial do sistema enquanto plataforma confiável e extensível (Semastin et al., 2018).

2.2 TRABALHOS RELACIONADOS

Nesta sessão são apresentados três trabalhos relacionados e um breve comparativo entre eles.

2.2.1 *Web-based Laboratory Management System - Federal Polytechnic, Níger*

No trabalho de Akeem Adekunle et al. (2024), o foco está na automação da reserva, rastreio e inventário de equipamentos de laboratório. Utilizando metodologia clássica (análise, *design*, implementação e testes), a solução proporciona painéis, gráficos e histórico de transações em tempo real. O *Web-based Laboratory Management System* (WBMLS) mostrou-se eficiente para monitoramento contínuo de *hardware*, reduzindo erros humanos e agilizando processos, assimilando resultados que corroboram a necessidade de sistemas organizacionais com interface acessível e controle centralizado, princípios que também embasam a presente proposta.

2.2.2 Cadets Administration System - Nusantara Maritime Academy

O trabalho de Fadilah et al. (2021) descreve a construção de um sistema web para simplificar processos manuais como registro de permissões e bolsas. A pesquisa adotou o modelo *waterfall* (Manhas, 2017) e ferramentas como PHP, CodeIgniter e MySQL. Os autores apontam que a solução desenvolvida diminuiu o tempo gasto em burocracias, substituindo o deslocamento presencial por uma plataforma *online* eficiente. Apesar das limitações do ciclo de desenvolvimento empregado (*waterfall*) e das escolhas tecnológicas de menor escalabilidade, o contexto e os resultados são especialmente relevantes. Tais sistemas representam a digitalização de práticas antes realizadas manualmente, um movimento que o sistema aqui desenvolvido também acompanhou. A diferença está na abordagem: enquanto esses projetos optam por implementações mais lineares, este trabalho aposta em uma estrutura iterativa e em tecnologias amplamente utilizadas na área de aplicações web, especialmente aquelas que favorecem a modularização e a escalabilidade das funcionalidades.

2.2.3 Avaliação qualitativa em laboratório clínico

Em um estudo conduzido na Universidade Federal Fluminense, avaliou a percepção da equipe sobre a implantação de um sistema *online* de gestão de laboratório clínico (Timoteo et al., 2020). Por meio de questionários e grupos focais, observou-se aumento na organização, na comunicação e na autonomia dos usuários, embora tenha sido identificado que um sistema desse tipo exige constante treinamento da equipe. Esses achados apoiam a hipótese frequentemente observada em sistemas web: a tecnologia sozinha não resolve, mas precisa vir acompanhada de práticas de adoção e suporte, aspecto relevante para futuros testes de usabilidade do sistema desenvolvido.

2.2.4 Comparativo dos trabalhos relacionados

Os três trabalhos apresentados oferecem um panorama consistente sobre a adoção de sistemas web no ambiente acadêmico, destacando os benefícios (eficiência, automação, comunicação) e as limitações (ciclos rígidos de desenvolvimento, necessidade de suporte humano).

O que diferencia a proposição deste trabalho é, sobretudo, sua abrangência; enquanto os trabalhos relacionados concentram-se em um único domínio, seja a administração de cadetes, a reserva e o rastreamento de equipamentos ou a avaliação de uso de uma plataforma externa, o sistema aqui proposto integra, em uma única aplicação, a gestão de membros da equipe, equipamentos, projetos de pesquisa e parcerias, aliada a uma página pública de divulgação do laboratório. Somam-se a isso a adoção de um processo de desenvolvimento iterativo e incremental, o uso de tecnologias modernas e amplamente difundidas (Django, React, TypeScript e PostgreSQL), o controle de acesso baseado em papéis e a disponibilização do código de forma

aberta, em uma arquitetura modular voltada especificamente ao ambiente de pesquisa e colaboração em laboratórios universitários. A Tabela 1 sintetiza essa comparação, incluindo o sistema proposto na última linha.

Tabela 1 – Comparativo entre os trabalhos relacionados e a solução proposta.

Estudo	Abordagem	Tecnologias	Benefícios
Cadets Administration System (Nusantara Maritime Academy)	Modelo em cascata (<i>Waterfall</i>)	PHP, CodeIgniter, MySQL	Digitalização de processos manuais como envio de permissões e registros; melhora na eficiência administrativa
WBLMS – Federal Polytechnic, Níger	Metodologia tradicional (análise, <i>design</i> e testes)	PHP, MySQL	Automação de reservas e rastreamento de equipamentos; visualização em tempo real e redução de erros humanos
Sistema de Gestão de Laboratório Clínico (UFF)	Avaliação qualitativa com usuários	Plataforma <i>online</i> (ex: Quartz)	Melhorias na organização e comunicação; porém, requer treinamento constante para adoção eficaz
Sistema proposto (este trabalho)	Desenvolvimento iterativo e incremental	Django, React, TypeScript, PostgreSQL	Gestão integrada e centralizada de membros da equipe, equipamentos, projetos e parcerias em uma única plataforma modular e de código aberto, com controle de acesso por papéis e suporte bilíngue, voltada a laboratórios de pesquisa e desenvolvimento

Fonte: o autor.

3 REQUISITOS DA APLICAÇÃO E MODELAGEM

Este capítulo apresenta os requisitos identificados para o sistema desenvolvido, descrevendo as funcionalidades implementadas e as necessidades que a aplicação atende. Em seguida, são introduzidos os principais elementos da modelagem do sistema, abordando a estrutura geral da aplicação, as tecnologias utilizadas e a forma como os componentes do frontend, backend e banco de dados se relacionam.

3.1 REQUISITOS DA APLICAÇÃO

A definição dos requisitos é uma etapa essencial no desenvolvimento de sistemas, pois orienta tanto o planejamento quanto a implementação da solução. Nesta seção, são descritas as funcionalidades que o sistema oferece, bem como as restrições e características técnicas observadas em sua construção. Os requisitos foram organizados em duas categorias: funcionais e não funcionais.

3.1.1 Requisitos Funcionais

Os requisitos funcionais descrevem o conjunto de funcionalidades que o sistema precisa fornecer para atender às demandas do laboratório. Os requisitos identificados e implementados estão listados a seguir:

- Permitir o cadastro, edição e exclusão de integrantes do laboratório, abrangendo professores (administradores), estudantes de graduação, mestrado, doutorado e outros colaboradores.
- Registrar materiais e equipamentos pertencentes ao laboratório, com identificação por meio de etiquetas ou códigos únicos.
- Exibir a lista de materiais cadastrados, com possibilidade de realizar buscas.
- Controlar o status de uso dos itens (disponível, em uso, manutenção, etc.), facilitando o acompanhamento de sua disponibilidade.
- Oferecer um sistema de autenticação de usuários, com acesso restrito a funcionalidades de acordo com o perfil de cada usuário.
- Permitir que os próprios usuários atualizem suas informações cadastrais.
- Disponibilizar diferentes níveis de permissão para acesso e uso da plataforma, sendo professor o cargo de administrador, e estudante e colaborador cargos com permissões voltadas à visualização.

- Gerar relatórios informativos com dados sobre uso de materiais, movimentações e estatísticas de participação dos membros.
- Funcionalidades de adição e gerenciamento de projetos de pesquisas, colaboradores, informações disponíveis na página principal como parcerias, contato, localização, etc.

Boa parte dos requisitos listados acima são relacionados à parte de gerenciamento de materiais e equipamentos do laboratório, incluindo seu uso e localização. Com base nessas informações, a geração de relatórios de inventário é uma tarefa prática e acessível. Outros requisitos estão voltados à parte de acesso público, como a apresentação dos projetos de pesquisa desenvolvidos no laboratório.

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais definem restrições técnicas, padrões de qualidade e condições operacionais que devem ser atendidas para garantir a eficiência, usabilidade e escalabilidade do sistema.

- O sistema deve ser acessível por meio de navegadores web modernos, sem a necessidade de instalação local.
- A interface do sistema deve ser responsiva, adaptando-se adequadamente a diferentes tamanhos de tela e dispositivos (como *notebooks*, *tablets* e *smartphones*).
- O *backend* da aplicação foi desenvolvido com o *framework* Django, utilizando a linguagem Python, e foi cuidadosamente documentado e versionado, a fim de facilitar futuras extensões e adaptações do sistema.
- O *frontend* foi implementado com React, garantindo uma experiência de navegação dinâmica e fluida.
- O banco de dados utilizado é relacional, com o sistema PostgreSQL responsável pelo armazenamento das informações.
- A autenticação dos usuários deve ser segura, com senhas armazenadas de forma criptografada.
- O código-fonte foi versionado utilizando Git e armazenado em repositório remoto no GitHub, garantindo rastreabilidade e organização do desenvolvimento. O repositório será mantido público, permitindo que qualquer interessado possa acessar, adaptar ou expandir a solução conforme suas próprias necessidades.
- A aplicação foi projetada para hospedagem em ambiente de nuvem, como AWS ou OCI, garantindo acesso remoto e escalabilidade da solução.

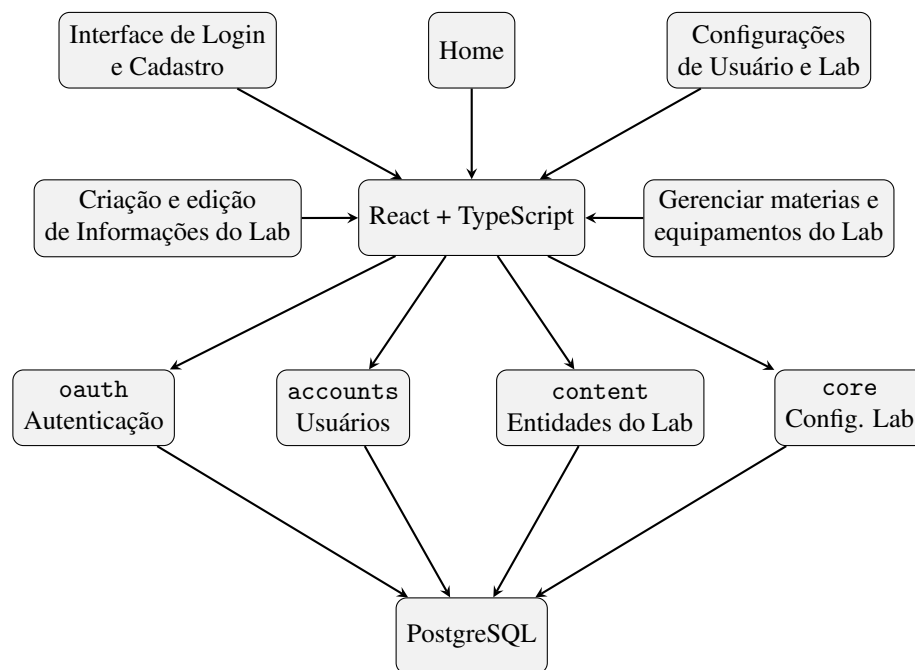
- O tempo de resposta para operações básicas (como cadastro, *login* ou consulta) deve ser suficientemente baixo, visando proporcionar uma boa experiência ao usuário.

3.2 MODELAGEM

A modelagem do sistema implementado reflete a organização modular e a comunicação entre as diferentes camadas da arquitetura do sistema, composta por *frontend*, *backend* e banco de dados. A separação clara entre essas camadas contribui para uma estrutura mais compreensível e de fácil manutenção, especialmente em projetos com múltiplas funções interativas e administrativas.

A Figura 4 apresenta o fluxo de interação entre essas camadas, evidenciando como as interfaces do *frontend* se comunicam com os diferentes *apps* do *backend* Django, que por sua vez, persistem os dados no PostgreSQL.

Figura 4 – Fluxo geral de interação entre as camadas do sistema.



Fonte: o autor.

No *frontend*, a biblioteca React, aliada ao Vite e à linguagem TypeScript, compõe a base da camada de apresentação. Essa combinação permite a criação de interfaces dinâmicas e seguras, o que oferece aos usuários uma navegação fluida. As páginas principais incluem as de *login* e criação de conta, painel administrativo para criação e edição de projetos de pesquisa, cadastro de materiais e equipamentos, além de ter páginas para configurações de perfis de usuários e informações do laboratório. Além disso, componentes reutilizáveis, como notificações, *uploads* de arquivos e a barra de navegação, centralizam funcionalidades comuns a diversas rotas da aplicação.

Por trás das interfaces, a lógica do servidor é gerenciada pelo *framework* Django. A estrutura do *backend* está organizada em múltiplos *apps*, cada um responsável por aspectos específicos do sistema. O *app accounts* gerencia os dados dos usuários e a aprovação de novos cadastros. O *content* concentra as entidades do domínio da aplicação, como equipamentos, projetos e parcerias. O *core* contém as configurações institucionais do laboratório. O *app oauth* é responsável por toda a lógica de autenticação e sincronização de contas, enquanto o *sitewide* concentra funções utilitárias e decoradores de acesso.

O armazenamento das informações é feito por meio do banco de dados PostgreSQL. Nele estão armazenadas as entidades relacionadas a usuários, configurações do laboratório, arquivos e demais dados essenciais à operação do sistema.

A integração entre *frontend* e *backend* ocorre por meio da aplicação servida com uWSGI (Unbit, 2014), garantindo a conexão entre a execução do servidor e a entrega dos arquivos da aplicação web. Essa arquitetura garante que as funcionalidades do sistema, como autenticação, cadastro, edição de informações e gestão de usuários, estejam devidamente interligadas entre a interface, o processamento e a persistência de dados.

4 DESENVOLVIMENTO

Este capítulo descreve o processo de desenvolvimento do sistema, detalhando as decisões de implementação adotadas ao longo do projeto, a organização dos módulos desenvolvidos e as funcionalidades entregues. O desenvolvimento seguiu uma abordagem iterativa, com entregas incrementais que permitiram validar e ajustar continuamente as escolhas de arquitetura e de tecnologias.

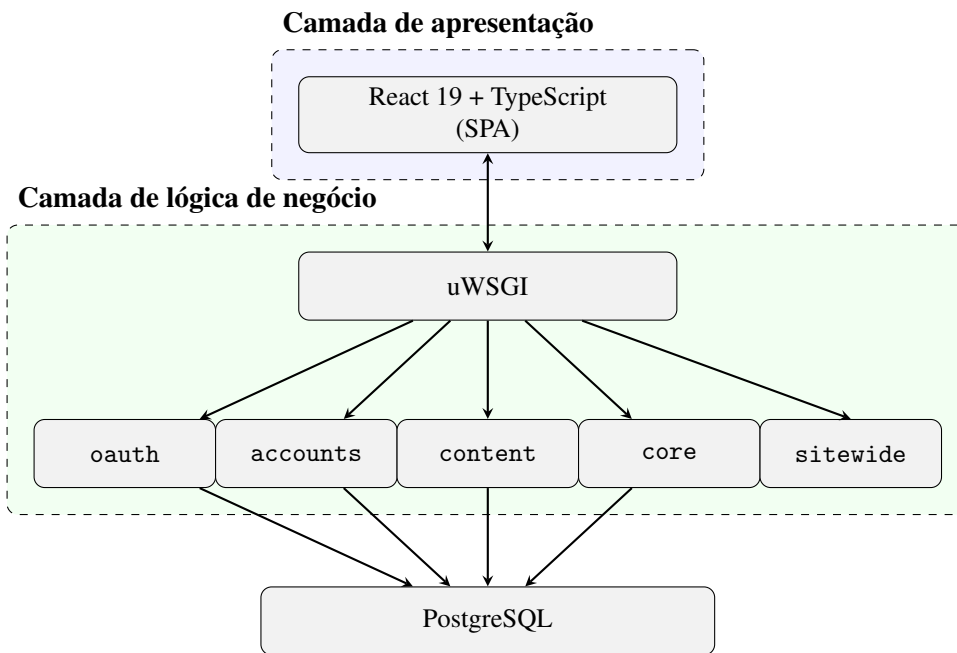
4.1 ARQUITETURA DO SISTEMA

A arquitetura implementada segue o modelo cliente-servidor, com separação clara entre a camada de apresentação e a camada de lógica de negócio. O *frontend*, construído com React 19 e TypeScript, é entregue ao navegador como uma Aplicação de Página Única – *Single Page Application* (SPA), aplicação de página única em que a navegação entre telas ocorre sem recarregamentos completos da página. O *backend*, desenvolvido com Django 6, expõe uma API que responde às requisições originadas pelo cliente.

A Figura 5 apresenta a estrutura geral da aplicação com seus componentes. A integração entre as duas camadas é gerenciada pelo servidor de aplicação uWSGI (Unbit, 2014). Em ambiente de desenvolvimento, o uWSGI inicializa tanto o processo Django quanto o servidor Vite como um processo *daemon* paralelo, habilitando a recarga automática de módulos a cada alteração no código-fonte. Em produção, o Vite gera um pacote estático dos arquivos do *frontend*, que é coletado pelo Django e servido como arquivos estáticos convencionais.

O *backend* foi organizado em cinco *apps* Django independentes, cada um responsável por um domínio funcional distinto do sistema.

Figura 5 – Arquitetura geral do sistema.



Fonte: o autor.

4.2 BACKEND

O *backend* da aplicação foi desenvolvido com o *framework* Django na versão 6, utilizando Python como linguagem de programação e o gerenciador de pacotes uv (Astral, 2024) para isolamento e controle do ambiente de execução. A organização em *apps* independentes segue o princípio de separação de responsabilidades, tornando cada módulo coeso e com escopo bem delimitado. As seções a seguir descrevem os principais *apps* desenvolvidos.

4.2.1 Módulo de contas (accounts)

O módulo `accounts` é responsável pelo gerenciamento de usuários e perfis. Um de seus aspectos centrais é o modelo de usuário personalizado, no qual o endereço de *e-mail* substitui o nome de usuário convencional como identificador único para autenticação. Essa escolha se adequa ao contexto acadêmico, em que o *e-mail* institucional já serve como referência de identidade natural dos membros do laboratório. Por conveniência, um nome de usuário único também pode ser definido no cadastro para facilitar *logins* futuros.

O modelo `User` armazena informações pessoais e profissionais, como nome, data de nascimento, telefone, *e-mail* de contato público e link para o currículo Lattes. Para atender ao caráter bilíngue do sistema, os campos de texto voltados ao público, como a biografia, são armazenados em pares com sufixos `_pt` e `_en`, permitindo que o conteúdo seja exibido em português ou inglês conforme a preferência do visitante.

O controle de acesso é definido por papéis armazenados diretamente no modelo como uma lista em formato JSON. Os papéis disponíveis são: `professor`, `student`, `collaborator` e `inventory_manager`. Essa abordagem dispensa o uso do sistema de grupos nativo do Django, oferecendo maior flexibilidade para a lógica de autorização específica do sistema, implementada por dois métodos auxiliares: `can_manage_all()` (que é exclusivo para professores e administradores dos sistema) e `can_manage_equipment()` (para professores, administradores e gerentes de inventário).

O módulo inclui também o modelo `Invitation`, que representa convites enviados por professores a novos membros. Cada convite é representado por um *token* único de 64 caracteres, com validade de sete dias. Quando um novo usuário acessa o sistema por meio de um convite válido, é direcionado a uma página de cadastro com os campos pré-preenchidos com as informações fornecidas pelo professor. Ao concluir o registro, a conta é criada já aprovada, sem necessidade de aprovação manual por parte de um professor.

4.2.2 Módulo de autenticação (oauth)

O módulo `oauth` centraliza a lógica de autenticação e gerenciamento de sessão da aplicação. A implementação utiliza sessões do Django combinadas com *tokens* Token Web JSON – *JSON Web Token* (JWT) (Auth0, 2015), gerados com o algoritmo HS256, para identificar e validar os usuários autenticados.

O módulo expõe os seguintes *endpoints*¹: entrada na plataforma (*sign-in*), criação de conta (*sign-up*), encerramento de sessão (*sign-out*), verificação de *e-mail*, sincronização do estado do usuário e consulta da sessão ativa. A sincronização é executada automaticamente pelo *frontend* a cada carregamento da aplicação, garantindo que as informações de perfil e permissões estejam sempre atualizadas na interface.

O fluxo padrão de cadastro inclui a verificação do endereço de *e-mail* e a aprovação manual por parte de um professor. Usuários convidados por meio do sistema de convites são direcionados a uma página de cadastro com os dados pré-preenchidos pelo professor e, ao concluir o registro, têm a conta aprovada automaticamente, sem necessidade de intervenção manual.

4.2.3 Módulo de conteúdo (content)

O módulo `content` é o núcleo do domínio da aplicação, agrupando os modelos que representam as entidades gerenciadas pelo laboratório. As principais entidades implementadas são:

- **ResearchArea**: áreas de atuação científica do laboratório, com título, descrição bilíngues e referência externa para mais informações.

¹Um *endpoint* é um ponto de acesso de uma API, identificado por uma URL à qual o cliente envia requisições para acessar um recurso ou executar uma operação.

- **Project:** projetos de pesquisa em desenvolvimento, com lista de membros participantes e suporte a ordenação personalizada.
- **Partnership:** parcerias institucionais, armazenando nome, logotipo e referência externa.
- **Equipment:** equipamentos do laboratório, com identificador único, estado de conservação, localização por sala e seção, e responsável pelo item.
- **Room e RoomSection:** representam, respectivamente, as salas físicas do laboratório e suas subdivisões, permitindo rastreamento granular da localização de cada equipamento.
- **EquipmentState e IdentificationCategory:** tabelas auxiliares para estados de conservação e categorias de equipamentos.

Todos os modelos implementam o método `export()`, que retorna um dicionário Python com os dados do objeto formatados para serialização em JSON. Essa abordagem foi adotada em lugar dos serializadores do Django REST Framework, oferecendo controle granular sobre quais campos e relações são incluídos em cada contexto de resposta da API.

O módulo contempla ainda a geração de relatórios nos formatos PDF e planilha (XLSX), implementada com as bibliotecas `reportlab` e `openpyxl`. Os relatórios consolidam informações sobre equipamentos e membros do laboratório, facilitando tarefas de auditoria e prestação de contas.

4.2.4 Módulo de configurações do laboratório (`core`)

O módulo `core` gerencia as configurações globais e a identidade visual do laboratório. O modelo central, `LabSettings`, segue o padrão de instância única, armazenando o nome do laboratório, logotipo, missão institucional nos dois idiomas, dados de contato, endereço físico e as cores personalizáveis da página principal.

O módulo inclui ainda o modelo `AboutImage`, que armazena as imagens exibidas na seção de apresentação pública do laboratório, com suporte a ordenação personalizada pelo administrador.

4.2.5 Utilitários transversais (`sitewide`)

O módulo `sitewide` concentra componentes utilitários utilizados pelos demais *apps*. O principal deles é o decorador `user_access_required`, que protege os *endpoints* da API contra acessos não autorizados. O decorador também intercepta exceções de validação e retorna respostas de erro padronizadas com o código de status HTTP 400 (*Bad Request*), que sinaliza ao cliente uma requisição malformada ou inválida, mantendo consistência nas respostas da API.

O módulo implementa também um sistema de identificadores públicos baseado na biblioteca `Hashids` (Vujic, 2022), que codifica os identificadores numéricos internos dos modelos

em cadeias de caracteres curtas e opacas. Esse mecanismo evita a exposição de sequências incrementais nas URLs da API, dificultando a enumeração de recursos por terceiros.

O conjunto de mecanismos descritos ao longo desta seção, como organização em *apps* com responsabilidades bem delimitadas, padrão de resposta JSON consistente via método `export()`, autorização baseada em papéis e ofuscação de identificadores, concretiza o **Objetivo Específico O4**, garantindo que a API seja segura, coerente e extensível a diferentes implementações de *frontend*.

4.3 FRONTEND

O *frontend* foi desenvolvido como uma SPA utilizando React 19 (React, 2024), TypeScript e Vite (VoidZero, 2024). A adoção do TypeScript em lugar do JavaScript puro contribui para a detecção precoce de erros de tipo durante o desenvolvimento, aumentando a confiabilidade da base de código. O Vite, por sua vez, oferece tempos de inicialização e recompilação significativamente menores em relação a ferramentas tradicionais de empacotamento, agilizando o ciclo de desenvolvimento.

4.3.1 Estrutura de rotas e controle de acesso

O roteamento da aplicação é gerenciado com o React Router e organizado em três categorias de acesso. As rotas públicas, como a página principal do laboratório e a tela de autenticação, são acessíveis sem que o usuário esteja autenticado. As rotas autenticadas exigem sessão ativa e são protegidas pelo componente `PrivateRoute`. As rotas administrativas aplicam verificações adicionais de papel de usuário, utilizando as funções utilitárias `canManageAll` (exclusiva para professores) e `canManageEquipment` (para professores e gerentes de inventário). Essas funções são definidas tanto no *frontend* quanto no *backend*, mantendo a consistência da lógica de autorização entre as duas camadas da aplicação.

4.3.2 Gerenciamento de estado global

O estado global da aplicação é gerenciado com a *Context API* do React, combinada a um *reducer* centralizado que controla os dados compartilhados entre componentes, incluindo o estado de autenticação do usuário e as coleções de dados carregadas da API.

A inicialização da sessão é coordenada pelo serviço `AuthHandler`, executado uma única vez durante a inicialização da aplicação. Esse serviço realiza uma requisição ao *endpoint* de sincronização do *backend* e restaura o estado do usuário, caso exista uma sessão ativa, antes que qualquer rota seja renderizada.

4.3.3 Sistema de modais

Uma janela modal é um elemento de interface que se sobrepõe ao conteúdo principal da aplicação e bloqueia a interação com o restante da página até que o usuário conclua ou cancele a ação, sendo comumente empregada em formulários, confirmações e notificações.

Para o gerenciamento de janelas modais, foi adotado um sistema de registro dinâmico de componentes, implementado na biblioteca interna `my-own-modal-handler`. Cada componente modal é registrado por nome na inicialização da aplicação e pode ser exibido a partir de qualquer ponto da árvore de componentes, sem a necessidade de propagar propriedades por múltiplos níveis de hierarquia. Essa abordagem reduz o acoplamento entre componentes e centraliza o ciclo de vida das janelas modais.

4.3.4 Internacionalização

A interface do sistema suporta dois idiomas: português e inglês. A internacionalização é implementada com a biblioteca `i18next`, que detecta automaticamente o idioma configurado no navegador do usuário e carrega as traduções correspondentes. Os textos estáticos da interface são armazenados em arquivos de localização separados por idioma, enquanto o conteúdo dinâmico, como títulos e descrições das entidades do laboratório, é recuperado diretamente dos campos bilíngues (`_pt/_en`) armazenados no banco de dados.

4.4 FUNCIONALIDADES IMPLEMENTADAS

Esta seção apresenta uma visão consolidada das funcionalidades entregues ao final do desenvolvimento. O conjunto de recursos implementados cobre os requisitos funcionais identificados no capítulo anterior.

Gerenciamento de membros: professores podem convidar novos integrantes com informações pré-preenchidas.

Ao clicar no link do convite, o novo membro é direcionado a uma página de cadastro já preenchida e, ao concluir o registro, entra aprovado automaticamente no sistema. Além disso, também podem remover membros existentes e editar funções, cargos e sala associada de cada integrante.

Com o desenvolvimento dessas funcionalidades, o **Objetivo Específico O1** foi plenamente atendido. A Figura 6 ilustra a interface de gerenciamento de membros.

Figura 6 – Interface de gerenciamento de membros do laboratório.

Usuários
Gerencie os usuários, cargos e salas do sistema

Buscar por nome ou e-mail...

Cargos
Estudante Professor

Salas
410 411 412

Usuários (4)

NOME	EMAIL	CARGOS	SALA	FUNÇÕES	STATUS
Gustavo Santos	gustavogds142000@gmail.com	Estudante	-	Professor	Ativo
Ismael Seidel	ismael.seidel@ufsc.br	Professor	411	Professor	Ativo
João Vitor	vipar99280@dosbee.com	Estudante	-	Student	Ativo
Temp Name	dowog52674@okcpress.com	Estudante	-	Student	Ex-membro

Fonte: o autor.

Gerenciamento de equipamentos e inventário: o sistema permite o cadastro completo de equipamentos com identificador único, categoria, estado de conservação, localização por sala e seção, e responsável. Professores e gerentes de inventário podem criar, editar e remover registros.

Com o desenvolvimento dessas funcionalidades, o **Objetivo Específico O2** foi plenamente atendido. A Figura 7 ilustra a interface de gestão de equipamentos.

Figura 7 – Interface de gestão de equipamentos do laboratório.

Equipamentos
Gerencie os materiais, equipamentos e salas do laboratório

Buscar por categoria, ID, nome ou usuário...

Categorias de Identificação
Equipamento Material

Estados dos Equipamentos
Novo Usado Quebrado

410

SEM SEÇÃO

ID	CATEGORIA	NOME	OBS.	RESPONSÁVEL	USUÁRIOS	ESTADO	STATUS
eqp_015	Equipamento	HD Externo 2 TB	Backup sema...	—	—	Novo	Ativo

MESA 001

ID	CATEGORIA	NOME	OBS.	RESPONSÁVEL	USUÁRIOS	ESTADO	STATUS
eqp_001	Equipamento	Monitor 27 polegadas Dell	—	—	—	Usado	Ativo
eqp_002	Equipamento	Notebook Lenovo ThinkPad E14	Uso compartil...	—	—	Usado	Ativo
eqp_016	Equipamento	Teclado Mecânico USB	—	—	—	Usado	Ativo
eqp_017	Equipamento	Mouse Óptico USB	—	—	—	Usado	Ativo

MESA 002

ID	CATEGORIA	NOME	OBS.	RESPONSÁVEL	USUÁRIOS	ESTADO	STATUS
eqp_01444	Equipamento	Servidor Dell PowerEdge	—	—	—	Usado	Ativo

MESA 003

Fonte: o autor.

Projetos de pesquisa e áreas de atuação: projetos em desenvolvimento são cadastrados com título bilíngue, descrição e lista de membros participantes. As áreas de atuação funcionam de forma análoga, agrupando os temas de investigação do laboratório. Projetos suportam ordenação personalizada dos membros por meio de uma ordem definida na seleção dos participantes desse projeto.

Parcerias institucionais: parceiros são cadastrados com nome, logotipo e referência externa, sendo exibidos na página pública do laboratório.

Configurações e identidade visual: professores podem personalizar as informações institucionais exibidas publicamente, incluindo nome, missão, dados de contato, endereço, link de mapa e as cores da página principal.

Sistema de convites: professores podem gerar convites para novos membros, com validade de sete dias. O link de convite pode incluir informações pré-preenchidas, como nome

e cargo, agilizando o processo de integração.

Geração de relatórios: o sistema oferece a exportação de relatórios consolidados em formato PDF e planilha (XLSX), reunindo informações sobre os recursos e membros do laboratório.

A incorporação iterativa de funcionalidades complementares ao longo do desenvolvimento, como relatórios, sistema de convites e configurações de identidade visual, concretiza o **Objetivo Específico O5**. A Figura 8 apresenta a interface de geração de relatórios.

Figura 8 – Interface de geração de relatórios.

← Voltar

Novo Relatório

Configure e gere um relatório com os dados do laboratório

Nome do Relatório *

Relatório Mensal Junho 2026

Formato *

PDF
Documento para impressão e visualização

Excel (XLSX)
Planilha para análise e edição de dados

Seções do Relatório *

Selecione seções e personalize as colunas que aparecerão em cada tabela.

☐ Áreas de Pesquisa

☒ **Projetos** 5 de 5 colunas ▾

Colunas da tabela Desmarcar Tudo

☒ ID ☒ Título ☒ Descrição ☒ Integrantes

☒ Status

☒ **Usuários** 6 de 6 colunas ▾

Colunas da tabela Desmarcar Tudo

☒ ID ☒ Nome ☒ E-mail ☒ Cargo

☒ Funções ☒ Sala

Agrupar por: ☐ Sala

☐ Equipamentos

☐ Parcerias

Cancelar Gerar Relatório

Fonte: o autor.

Página pública: a página inicial apresenta as informações públicas do laboratório, membros, projetos, áreas de pesquisa, parcerias e informações de contato, de forma responsiva e adaptada a diferentes tamanhos de tela.

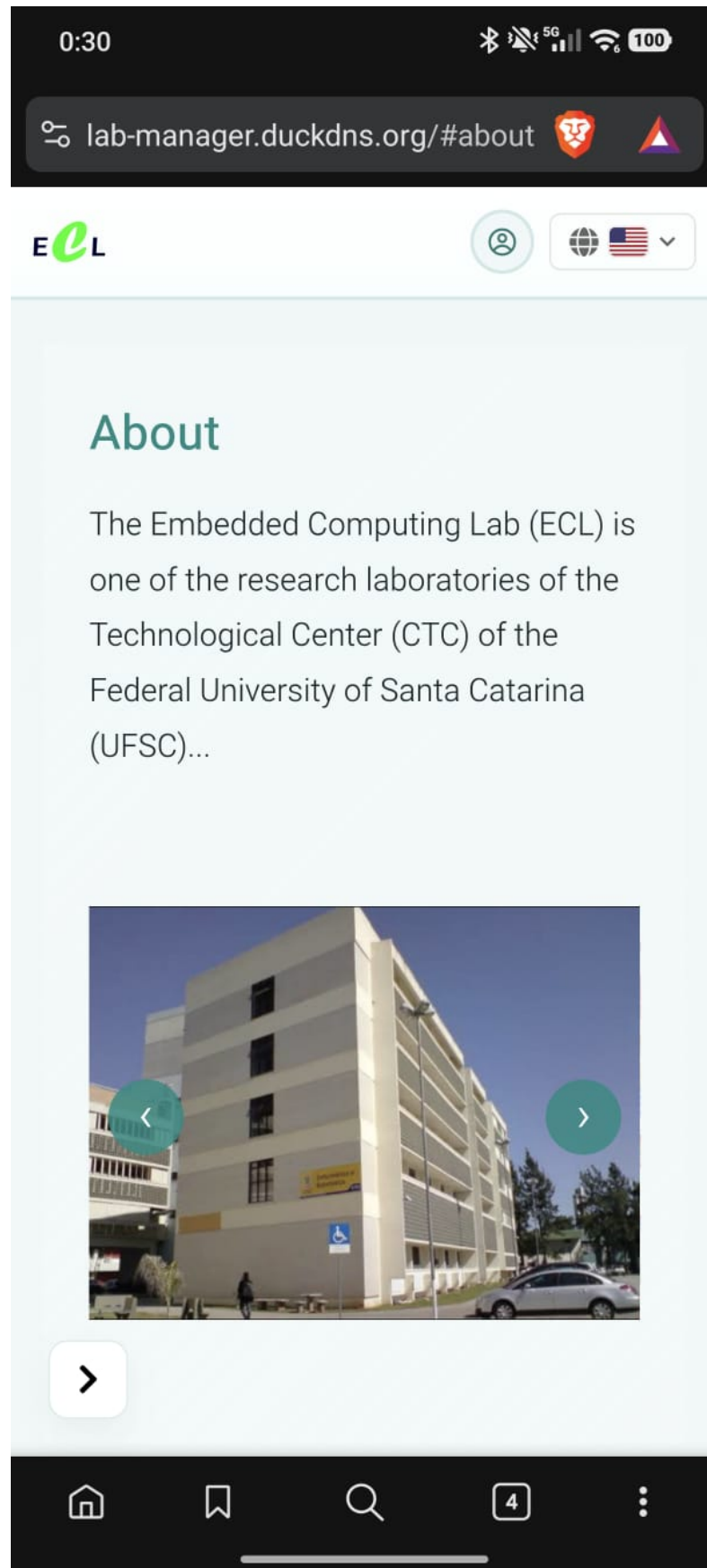
O uso de técnicas de *layout* responsivo na construção da interface, tanto na página pública quanto nos módulos de gestão interna, resultou em uma experiência de uso consistente e intuitiva em diferentes dispositivos, concretizando os **Objetivos Específicos O3 e O6**. As Figuras 9 e 10 ilustram, respectivamente, a página pública acessada em um navegador *desktop* e em um dispositivo móvel.

Figura 9 – Página pública do laboratório.



Fonte: o autor.

Figura 10 – Página pública do laboratório acessada em dispositivo móvel.



Fonte: o autor.

4.5 DISTRIBUIÇÃO E IMPLANTAÇÃO

O sistema foi desenvolvido como *software* livre e de código aberto, com o código-fonte disponibilizado publicamente nos seguintes repositórios:

- <https://github.com/gustavogds/lab-manager>
- <https://codigos.ufsc.br/gustavo.gs/lab-manager>

Para execução local em ambiente de desenvolvimento, as instruções de configuração do ambiente, instalação de dependências e inicialização do servidor estão descritas no Apêndice B.

A implantação do sistema em um servidor de produção pode ser realizada seguindo o procedimento detalhado no Apêndice C, que cobre a configuração de todos os componentes da pilha tecnológica: banco de dados PostgreSQL, servidor de aplicação uWSGI, servidor *web* Nginx e certificado SSL com Let's Encrypt.

5 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um sistema web voltado ao gerenciamento compartilhado de laboratórios de pesquisa e desenvolvimento em ambientes acadêmicos. A motivação central partiu de um problema recorrente nesses espaços: a ausência de ferramentas integradas e acessíveis para organizar integrantes, equipamentos, projetos de pesquisa, parcerias e demais informações institucionais. A solução proposta e implementada visa suprir essa lacuna por meio de uma plataforma centralizada, acessível por navegador e adaptada às diferentes categorias de usuários que compõem um laboratório acadêmico.

O desenvolvimento do sistema foi conduzido com Django 6 no *backend*, React 19 e TypeScript no *frontend*, e PostgreSQL como banco de dados, conforme planejado. A organização do *backend* em *apps* independentes, *accounts*, *oauth*, *content*, *core* e *sitewide*, mostrou-se adequada para separar responsabilidades e manter a coesão de cada módulo ao longo do desenvolvimento. O *frontend*, estruturado como uma SPA, oferece uma experiência de navegação fluida e adaptada a diferentes dispositivos.

Entre as funcionalidades entregues, destacam-se: o gerenciamento completo de membros com controle de acesso por papéis; o inventário de equipamentos com rastreamento de localização, estado e responsável; o cadastro de projetos de pesquisa, áreas de atuação e parcerias; a personalização da identidade visual do laboratório; o sistema de convites para integração ágil de novos membros; e a exportação de relatórios em PDF e planilha. O suporte bilíngue em português e inglês, implementado tanto nos campos do banco de dados quanto na interface, amplia o alcance do sistema e o torna mais adequado a laboratórios com integrantes internacionais.

No que diz respeito às decisões técnicas, o armazenamento dos papéis de usuário como lista em formato JSON no modelo de usuário, em lugar do sistema de grupos nativo do Django, mostrou-se uma escolha prática e de fácil manutenção para as necessidades do sistema. Da mesma forma, a adoção do método `export()` nos modelos, em lugar de serializadores do Django REST Framework, ofereceu controle direto sobre a estrutura das respostas da API sem introduzir camadas adicionais de abstração.

Como trabalhos futuros, identificam-se algumas direções relevantes para a evolução do sistema. A implantação em um ambiente de produção com acesso real de usuários do laboratório permitiria coletar evidências empíricas sobre a usabilidade e a aderência da solução às necessidades reais do contexto. A realização de testes formais com os usuários finais, como professores e estudantes, possibilitaria identificar pontos de melhoria na interface e no fluxo de tarefas. Além disso, funcionalidades como notificações, reserva de equipamentos e integração com sistemas de autenticação institucionais, como o SSO da UFSC (UFSC, 2024), poderiam expandir a utilidade da plataforma.

Em síntese, o sistema desenvolvido atinge os objetivos propostos no início do trabalho, entregando uma aplicação funcional e documentada para o gerenciamento de laboratórios aca-

dêmicos. O código-fonte está disponível publicamente no repositório GitHub¹ e códigos.ufsc² do projeto, de modo que qualquer laboratório interessado possa adaptar ou estender a solução conforme suas próprias necessidades.

¹<https://github.com/gustavogds/lab-manager>

²<https://codigos.ufsc.br/gustavo.gs/lab-manager>

REFERÊNCIAS

- AKEEM ADEKUNLE, Abiona et al. **Design and Implementation of a Web based Laboratory Management System (WBLMS) (A Case Study of Federal Polytechnic, Ile – Oluji, Ondo State)**. [S.l.], nov. 2024. P. 15–20. Disponível em: <<https://jaaionline.org/archives/volume1/number2/design-and-implementation-of-a-web-based-laboratory-management-system-wblms-a-case-study-of-federal-polytechnic-ile-oluji-ondo-state/>>.
- ASTRAL. **uv — An extremely fast Python package manager**. Versão 0.5. [S.l.], 2024. Acessado em: 2025-07-03 | Disponível em: <https://docs.astral.sh/uv>.
- AUTH0, Inc. **JSON Web Tokens — Introduction**. [S.l.], 2015. Padrão definido pela RFC 7519. Acessado em: 2025-07-03 | Disponível em: <https://jwt.io/introduction>.
- CHIEU, Trieu C. et al. **Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment**. [S.l.], 2009. P. 281–286.
- DJANGO, Software Foundation. **Django Documentation**. Versão 6.0. [S.l.], 2025. Acessado em: 2025-07-03 | Disponível em: <https://docs.djangoproject.com>.
- ELECTRONIC FRONTIER FOUNDATION, EFF. **Certbot**. Versão 5.6.0. [S.l.], 2026. Acessado em: 2026-06-10 | Disponível em: <https://eff-certbot.readthedocs.io/en/stable/>.
- FADILAH, Ana Umul et al. **Design of Cadets Administration System for Nusantara Cilacap Maritime Academy Based On Website**. [S.l.], 2021. P. 142–147.
- GAROUSI, Vahid et al. A systematic mapping study of web application testing. **Information and Software Technology**, v. 55, n. 8, p. 1374–1396, 2013. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0950584913000396>>.
- MANHAS, Jatinder. Initial framework for website design and development. **International Journal of Information Technology**, v. 9, n. 4, p. 363–375, dez. 2017. Disponível em: <<https://doi.org/10.1007/s41870-017-0045-4>>.
- MOLINA-RÍOS, Jimmy; PEDREIRA-SOUTO, Nieves. Comparison of development methodologies in web applications. **Information and Software Technology**, v. 119, p. 106238, 2020. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0950584919302551>>.
- NGINX. **Nginx ("engine x")**. Versão 1.30. [S.l.], 2026. Acessado em: 2026-06-10 | Disponível em: <https://nginx.org/en/docs/>.
- OWASP FOUNDATION. **OWASP Top 10:2021 – The Ten Most Critical Web Application Security Risks**. [S.l.], 2021. Acessado em: 2026-06-13 | Disponível em: <https://owasp.org/Top10/>.

POSTGRESQL, Global Development Group. **PostgreSQL Documentation**. Versão 17. [S.l.], 2024. Disponível em: <https://www.postgresql.org/docs/>.

REACT, Team. **React Documentation**. Versão 19.0. [S.l.], 2024. Disponível em: <https://reactjs.org/docs/getting-started.html>.

ROSADO, Nuno et al. **Web application for academic management — A case study**. [S.l.], jun. 2016. P. 1–7.

AL-SALEM, Lana S.; ABU SAMAHA, Ala. Eliciting Web application requirements – an industrial case study. **Journal of Systems and Software**, v. 80, n. 3, p. 294–313, 2007.

Disponível em:

<<https://www.sciencedirect.com/science/article/pii/S016412120600149X>>.

SEMASTIN, Emil et al. Preventive Measures for Cross Site Request Forgery Attacks on Web-based Applications. **International Journal of Engineering and Technology(UAE)**, v. 7, p. 130–134, out. 2018.

TANENBAUM, Andrew S.; VAN STEEN, Maarten. **Distributed Systems: Principles and Paradigms**. 2. ed. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2007.

TIMOTEO, Margareth O. et al. The impact of online management systems: a qualitative assessment of staff perception at a clinical research laboratory. **Research, Society and Development**, v. 9, n. 10, out. 2020. Disponível em:

<<https://rsdjournal.org/index.php/rsd/article/view/9188>>.

UFSC. **Autenticação centralizada UFSC**. [S.l.], 2024. Acessado em: 2026-06-10 | Disponível em:

<https://servicosti.sistemas.ufsc.br/publico/detalhes.xhtml?servico=355>.

UNBIT. **uWSGI Application Server**. Versão 2.0. [S.l.], 2014. Acessado em: 2025-07-03 | Disponível em: <https://uwsgi-docs.readthedocs.io/>.

VINCENT, W.S. **Django for Professionals**. [S.l.]: Independently Published, 2022. Disponível em: <<https://books.google.com.br/books?id=0uqjDwAAQBAJ>>.

VOIDZERO, Inc. **Vite Documentation**. Versão 6.0. [S.l.], 2024. Disponível em:

<https://vite.dev/guide/>.

VUJIC, Ivan. **Hashids — generate short unique IDs from integers**. Versão 1.3.1. [S.l.], 2022. Acessado em: 2025-07-03 | Disponível em: <https://hashids.org>.

WU, Liupengfei et al. Strengths and weaknesses of client-server and peer-to-peer network models in construction projects. **International Journal of Construction Management**, Taylor & Francis, v. 24, n. 12, p. 1349–1363, 2024. eprint:

<https://doi.org/10.1080/15623599.2023.2185950>.

Apêndices

APÊNDICE A – ARTIGO

Sistema Web para Gerenciamento Compartilhado de um Laboratório de Pesquisa e Desenvolvimento

Gustavo Gonçalves dos Santos¹

¹Departamento de Informática e Estatística (INE), Centro Tecnológico
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, SC, Brasil

`gustavo.gs@grad.ufsc.br`

Abstract. *The efficient management of academic research and development laboratories involves challenges related to organizing resources and coordinating heterogeneous teams. Manual methods or non-integrated tools often lead to inadequate control over equipment, materials and members. This work presents a web system for the shared management of such laboratories, integrating, in a single platform, the control of members, the inventory of equipment identified by unique code labels, research projects, institutional partnerships and a public presentation page. The solution was built with a modern and widely adopted stack (Django, React, TypeScript and PostgreSQL), with role-based access control and open-source code. The results include improved control over resources and greater transparency in laboratory operations.*

Resumo. *O gerenciamento eficiente de laboratórios acadêmicos de pesquisa e desenvolvimento envolve desafios de organização de recursos e de coordenação de equipes heterogêneas. Métodos manuais ou ferramentas não integradas costumam levar à falta de controle sobre equipamentos, materiais e integrantes. Este trabalho apresenta um sistema web para o gerenciamento compartilhado desses laboratórios, integrando, em uma única plataforma, o controle de integrantes, o inventário de equipamentos identificados por etiquetas de código, projetos de pesquisa, parcerias institucionais e uma página pública de divulgação. A solução foi construída com um conjunto de tecnologias moderno e amplamente adotado (Django, React, TypeScript e PostgreSQL), com controle de acesso baseado em papéis e código aberto. Os resultados incluem a melhoria no controle dos recursos e maior transparência nas operações do laboratório.*

1. Introdução

O gerenciamento eficiente de laboratórios de pesquisa e desenvolvimento é cada vez mais relevante no contexto universitário¹. Além das tarefas de pesquisa, esses laboratórios frequentemente enfrentam desafios relacionados à organização e ao controle de materiais e equipamentos, bem como à administração de equipes compostas por integrantes de diferentes níveis de formação, como graduandos, mestrandos, doutorandos, técnicos e professores [Akeem Adekunle et al. 2024, Timoteo et al. 2020].

¹Artigo produzido como resumo do Trabalho de Conclusão de Curso em Ciências da Computação da UFSC, desenvolvido sob orientação do Prof. Ismael Seidel, Dr. (INE/UFSC).

Métodos tradicionais, de planilhas digitais a registros manuais em papel, muitas vezes resultam em processos pouco eficientes e em falta de controle adequado sobre os recursos disponíveis [Fadilah et al. 2021, Akeem Adekunle et al. 2024]. Isso pode comprometer tanto a produtividade quanto a transparência na utilização dos recursos do laboratório. Nesse contexto, soluções digitais baseadas na web vêm ganhando destaque como alternativa promissora, oferecendo acessibilidade remota, interfaces mais intuitivas e a possibilidade de reunir múltiplas funcionalidades em um só lugar [Rosado et al. 2016]. Tecnologias modernas costumam oferecer, ainda, bom nível de flexibilidade e escalabilidade, o que facilita futuras adaptações e expansões do sistema à medida que novas necessidades surgem.

O objetivo geral deste trabalho foi desenvolver um sistema web para o gerenciamento compartilhado de laboratórios de pesquisa e desenvolvimento, promovendo uma gestão eficiente, colaborativa e centralizada de integrantes, materiais e recursos, com foco na melhoria da organização, do controle e da transparência das operações laboratoriais. O sistema serve também como fonte de dados para a apresentação do laboratório por meio de um site público. Esse objetivo foi desdobrado em objetivos específicos: (O1) permitir o cadastro, a edição e a visualização de informações sobre os integrantes; (O2) implementar uma gestão de materiais com etiquetas de código para identificar e rastrear equipamentos; (O3) criar uma interface responsiva e intuitiva; (O4) desenvolver interfaces de programação de aplicações (APIs) robustas, seguras e documentadas; (O5) incorporar funcionalidades adicionais conforme as demandas surgissem durante o desenvolvimento; e (O6) garantir o funcionamento colaborativo em diferentes dispositivos e plataformas.

A principal contribuição do trabalho é a integração, em uma única aplicação, da gestão de membros da equipe, de equipamentos, de projetos de pesquisa e de parcerias, aliada a uma página pública de divulgação do laboratório. A solução adota tecnologias modernas e amplamente difundidas, controle de acesso baseado em papéis e disponibilização do código de forma aberta, em uma arquitetura modular voltada especificamente ao ambiente de pesquisa e colaboração em laboratórios universitários. O restante do artigo está organizado como segue. A Seção 2 apresenta os conceitos fundamentais e os trabalhos relacionados. A Seção 3 descreve os requisitos e a arquitetura do sistema. A Seção 4 detalha o desenvolvimento e as funcionalidades entregues. A Seção 5 apresenta as considerações finais e os trabalhos futuros.

2. Conceitos fundamentais e trabalhos relacionados

2.1. Sistemas web e modelo cliente-servidor

Um sistema web pode ser entendido como um tipo de aplicação que opera em rede e é acessada por navegadores, muitas vezes estruturada sobre arquiteturas distribuídas e apoiada em múltiplas camadas. O desenvolvimento de aplicações web, assim como o de sistemas tradicionais, exige processos complexos que envolvem coleta de requisitos e o uso de múltiplas linguagens e tecnologias, o que leva a considerável heterogeneidade técnica e arquitetural [Garousi et al. 2013]. De maneira geral, uma aplicação web é composta por duas partes principais: uma interface acessível via navegador (*frontend*), na qual os usuários realizam suas interações, e um servidor de aplicação (*backend*), responsável por processar requisições, aplicar regras de negócio, realizar autenticações e acessar os dados armazenados. Essas aplicações consolidaram-se como ferramentas fundamentais em

setores como comércio eletrônico, serviços bancários e serviços acadêmicos, justamente pela capacidade de oferecer acesso remoto, interação em tempo real e automatização de processos [Molina-Ríos and Pedreira-Souto 2020].

Entre as características mais marcantes desses sistemas estão os requisitos frequentemente voláteis, a presença de múltiplos perfis de usuários e a complexidade de conteúdo e estrutura, aliadas a um ciclo de desenvolvimento tipicamente acelerado [Al-Salem and Abu Samaha 2007]. Em contextos institucionais, como o de universidades, essas características tornam os sistemas web particularmente úteis para lidar com fluxos informacionais diversos, demandas de acesso remoto e a necessidade de colaboração entre usuários com diferentes níveis de envolvimento. Quanto à hospedagem, embora a computação em nuvem tenha se consolidado como o principal meio de escalonamento de aplicações web [Chieu et al. 2009], ainda há instituições que preferem manter parte de suas soluções em servidores locais, por questões de segurança, política interna ou limitações orçamentárias.

Por trás de uma interação em uma página web existe uma estrutura de comunicação que há décadas serve de base para boa parte das aplicações distribuídas, o modelo cliente-servidor. Ele parte de um princípio direto: uma entidade (o cliente) faz uma solicitação, enquanto outra (o servidor) responde. Esse equilíbrio de papéis carrega implicações concretas em escalabilidade, segurança e controle de fluxo [Wu et al. 2024]. Apesar do surgimento de arquiteturas mais descentralizadas, o modelo permanece uma estrutura sólida para aplicações que exigem consistência e centralização [Tanenbaum and Van Steen 2007]. Sua compatibilidade com os paradigmas modernos de desenvolvimento web também explica sua persistência: *frameworks* como Django seguem esse padrão ao separar as requisições HTTP do processamento e da renderização realizados no servidor, e bibliotecas de *frontend* como React consomem rotas e dados providos por serviços hospedados em servidores.

2.2. Autenticação

Dentre as funcionalidades que compõem um sistema web, poucas são tão críticas quanto os mecanismos de autenticação [OWASP Foundation 2021]. A autenticação consiste no processo de identificar e validar um usuário com base em algum tipo de credencial, geralmente combinando identificador (como endereço de *e-mail* ou nome de usuário) e senha. Com a sofisticação dos ataques, esse processo tem evoluído para incorporar camadas adicionais, como autenticação em dois fatores, *tokens* temporários e autenticação via provedores externos. No sistema desenvolvido, a autenticação segue um fluxo com *e-mail* e senha, armazenados de forma segura, com uso de *tokens* de proteção contra falsificação de requisições entre sítios (CSRF). A camada de autenticação foi mantida flexível, de modo a possibilitar a adoção futura de outros métodos, como a integração com sistemas de autenticação centralizada já utilizados pela UFSC [UFSC 2024, Semastin et al. 2018].

2.3. Trabalhos relacionados

Um dos trabalhos relacionados concentra-se na automação da reserva, do rastreamento e do inventário de equipamentos de laboratório [Akeem Adekunle et al. 2024]. Utilizando metodologia clássica de análise, projeto, implementação e testes, a solução oferece painéis, gráficos e histórico de transações em tempo real, mostrando-se eficiente

para o monitoramento contínuo de *hardware* e para a redução de erros humanos. Outro trabalho descreve a construção de um sistema web para simplificar processos manuais, como registro de permissões e bolsas, adotando o modelo em cascata (*waterfall*) [Fadilah et al. 2021, Manhas 2017] e tecnologias como PHP, CodeIgniter e MySQL; a solução diminuiu o tempo gasto em burocracias, substituindo o deslocamento presencial por uma plataforma *online*. Por fim, um estudo conduzido na Universidade Federal Fluminense avaliou a percepção da equipe sobre a implantação de um sistema *online* de gestão de laboratório clínico [Timoteo et al. 2020]; observou-se aumento na organização, na comunicação e na autonomia dos usuários, embora tenha sido identificado que um sistema desse tipo exige treinamento constante da equipe.

Os três trabalhos oferecem um panorama consistente sobre a adoção de sistemas web no ambiente acadêmico, destacando benefícios (eficiência, automação, comunicação) e limitações (ciclos rígidos de desenvolvimento e necessidade de suporte humano). O que diferencia a proposta deste trabalho é, sobretudo, sua abrangência: enquanto os trabalhos relacionados concentram-se em um único domínio, o sistema aqui proposto integra, em uma única aplicação, a gestão de membros da equipe, equipamentos, projetos de pesquisa e parcerias, aliada a uma página pública de divulgação. Somam-se a isso a adoção de um processo de desenvolvimento iterativo e incremental, o uso de tecnologias modernas e difundidas, o controle de acesso baseado em papéis e a disponibilização do código de forma aberta. A Tabela 1 sintetiza essa comparação, incluindo o sistema proposto na última linha.

Tabela 1. Comparativo entre os trabalhos relacionados e a solução proposta.

Estudo	Abordagem	Tecnologias	Benefícios
Cadets Administration System (Nusantara)	Modelo em cascata	PHP, CodeIgniter, MySQL	Digitalização de processos manuais, como envio de permissões e registros; melhora na eficiência administrativa
WBLMS (Federal Polytechnic, Níger)	Metodologia tradicional (análise, projeto e testes)	PHP, MySQL	Automação de reservas e rastreamento de equipamentos; visualização em tempo real e redução de erros humanos
Gestão de Laboratório Clínico (UFF)	Avaliação qualitativa com usuários	Plataforma <i>online</i>	Melhorias na organização e comunicação; requer treinamento constante para adoção eficaz
Sistema proposto (este trabalho)	Desenvolvimento iterativo e incremental	Django, React, TypeScript, PostgreSQL	Gestão integrada e centralizada de membros, equipamentos, projetos e parcerias em uma única plataforma modular e de código aberto, com controle de acesso por papéis e suporte bilíngue

3. Requisitos e arquitetura

Os requisitos foram organizados em funcionais e não funcionais. Entre os requisitos funcionais destacam-se: permitir o cadastro, a edição e a exclusão de integrantes, abrangendo professores (administradores), estudantes de graduação, mestrado e doutorado e demais colaboradores; registrar materiais e equipamentos com identificação por etiquetas

ou códigos únicos; exibir a lista de materiais cadastrados, com busca; controlar o status de uso dos itens (disponível, em uso, manutenção); oferecer autenticação de usuários com acesso restrito conforme o perfil; disponibilizar diferentes níveis de permissão, sendo professor o cargo de administrador; gerar relatórios sobre uso de materiais e participação dos membros; e gerenciar projetos de pesquisa, colaboradores e informações da página principal, como parcerias, contato e localização.

Entre os requisitos não funcionais destacam-se: acessibilidade por navegadores web modernos, sem instalação local; interface responsiva, adaptada a diferentes tamanhos de tela; *backend* em Django com a linguagem Python, documentado e versionado; *frontend* em React; banco de dados relacional PostgreSQL; autenticação segura, com senhas armazenadas de forma criptografada; versionamento com Git em repositório remoto público; projeto voltado a hospedagem em nuvem ou servidor local; e tempo de resposta baixo para operações básicas, como cadastro, *login* e consulta.

A arquitetura implementada segue o modelo cliente-servidor, com separação clara entre a camada de apresentação e a camada de lógica de negócio. O *frontend*, construído com React 19 e TypeScript, é entregue ao navegador como uma aplicação de página única (*Single Page Application*, SPA), em que a navegação entre telas ocorre sem recarregamentos completos da página. O *backend*, desenvolvido com Django 6, expõe uma API que responde às requisições do cliente e persiste os dados no PostgreSQL. A integração entre as camadas é gerenciada pelo servidor de aplicação uWSGI [Unbit 2014]: em desenvolvimento, ele inicializa tanto o processo Django quanto o servidor Vite como processo paralelo, habilitando a recarga automática de módulos; em produção, o Vite gera um pacote estático do *frontend*, coletado e servido pelo Django. A Figura 1 apresenta a estrutura geral da aplicação, com o *backend* organizado em cinco *apps* Django independentes, cada um responsável por um domínio funcional distinto.

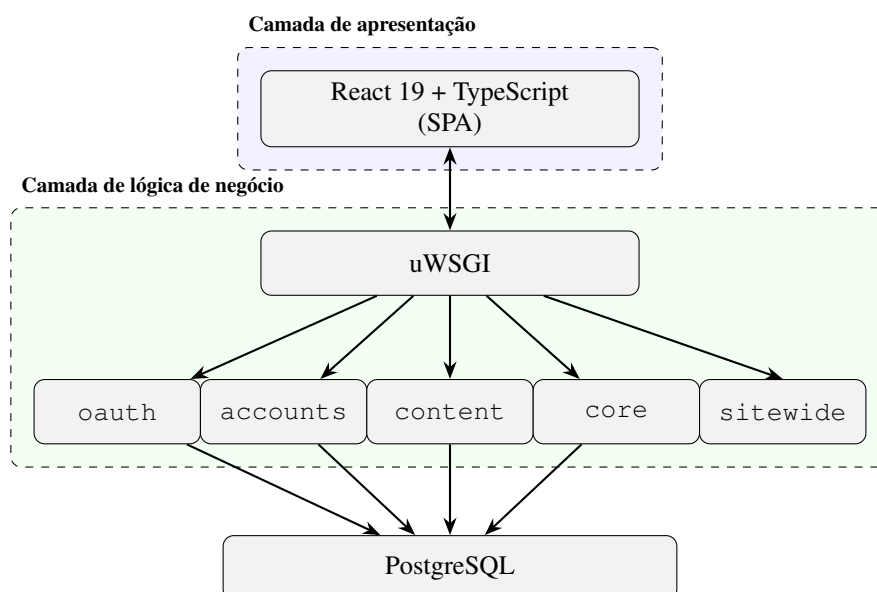


Figura 1. Arquitetura geral do sistema.

4. Desenvolvimento e funcionalidades

O desenvolvimento seguiu uma abordagem iterativa, com entregas incrementais que permitiram validar e ajustar continuamente as escolhas de arquitetura e de tecnologias. O código-fonte foi versionado com Git em repositório remoto público, e testes unitários e de integração foram conduzidos ao longo do projeto para identificar falhas e garantir a estabilidade do sistema.

4.1. Backend

O *backend* foi desenvolvido com o *framework* Django na versão 6 [Django Software Foundation 2025], utilizando Python como linguagem e o gerenciador de pacotes `uv` [Astral 2024] para isolamento do ambiente de execução. A organização em *apps* independentes segue o princípio de separação de responsabilidades, tornando cada módulo coeso e com escopo bem delimitado. A escolha de Python decorre também de sua ampla adoção nos cenários acadêmico e corporativo [Vincent 2022].

O *app* `accounts` é responsável pelo gerenciamento de usuários e perfis. Um de seus aspectos centrais é o modelo de usuário personalizado, no qual o endereço de *e-mail* substitui o nome de usuário convencional como identificador único, escolha adequada ao contexto acadêmico. O modelo armazena informações pessoais e profissionais e, para atender ao caráter bilíngue do sistema, mantém os campos de texto voltados ao público em pares com sufixos `_pt` e `_en`. O controle de acesso é definido por papéis armazenados como uma lista em formato JSON (`professor`, `student`, `collaborator` e `inventory_manager`), o que dispensa o sistema de grupos nativo do Django e oferece maior flexibilidade para a lógica de autorização, implementada por métodos auxiliares como `can_manage_all()` e `can_manage_equipment()`. O módulo inclui ainda o modelo `Invitation`, que representa convites enviados por professores a novos membros por meio de um *token* único, com validade de sete dias; ao concluir o registro por um convite válido, a conta é criada já aprovada.

O *app* `oauth` centraliza a autenticação e o gerenciamento de sessão, combinando sessões do Django com *tokens* do tipo *JSON Web Token* (JWT), gerados com o algoritmo HS256 [Auth0, Inc. 2015]. O módulo expõe *endpoints* para entrada na plataforma, criação de conta, encerramento de sessão, verificação de *e-mail*, sincronização do estado do usuário e consulta da sessão ativa. A sincronização é executada pelo *frontend* a cada carregamento da aplicação, mantendo perfil e permissões sempre atualizados na interface.

O *app* `content` é o núcleo do domínio da aplicação e agrupa os modelos das entidades gerenciadas pelo laboratório: áreas de atuação científica (`ResearchArea`), projetos de pesquisa (`Project`), parcerias institucionais (`Partnership`), equipamentos (`Equipment`) e salas e seções (`Room` e `RoomSection`), que permitem o rastreamento granular da localização de cada equipamento. Todos os modelos implementam o método `export()`, que retorna um dicionário Python formatado para serialização em JSON. Essa abordagem foi adotada em lugar dos serializadores do Django REST Framework, oferecendo controle granular sobre quais campos e relações são incluídos em cada resposta da API. O módulo contempla ainda a geração de relatórios em PDF e planilha (XLSX), com as bibliotecas `reportlab` e `openpyxl`, consolidando informações sobre equipamentos e membros.

O *app core* gerencia as configurações globais e a identidade visual do laboratório por meio do modelo `LabSettings`, que segue o padrão de instância única e armazena nome, logotipo, missão nos dois idiomas, dados de contato, endereço e as cores personalizáveis da página principal. Por fim, o *app sitewide* concentra utilitários transversais, com destaque para o decorador `user_access_required`, que protege os *endpoints* contra acessos não autorizados e padroniza as respostas de erro, e para um sistema de identificadores públicos baseado na biblioteca `Hashids` [Vujic 2022], que codifica os identificadores numéricos internos em cadeias curtas e opacas, evitando a exposição de sequências incrementais nas URLs da API. O conjunto desses mecanismos concretiza o objetivo específico O4, ao garantir que a API seja segura, coerente e extensível a diferentes implementações de *frontend*.

4.2. Frontend

O *frontend* foi desenvolvido como uma SPA utilizando React 19 [React Team 2024], TypeScript e Vite [VoidZero, Inc. 2024]. A adoção do TypeScript contribui para a detecção precoce de erros de tipo durante o desenvolvimento, enquanto o Vite oferece tempos de inicialização e recompilação reduzidos, agilizando o ciclo de trabalho. O roteamento é organizado em três categorias de acesso: rotas públicas, acessíveis sem autenticação; rotas autenticadas, protegidas pelo componente `PrivateRoute`; e rotas administrativas, que aplicam verificações adicionais de papel por meio das funções `canManageAll` e `canManageEquipment`. Essas funções são definidas tanto no *frontend* quanto no *backend*, mantendo a consistência da lógica de autorização entre as duas camadas.

O estado global é gerenciado com a *Context API* do React combinada a um *reducer* centralizado, que controla os dados compartilhados entre componentes, incluindo o estado de autenticação e as coleções de dados carregadas da API. A inicialização da sessão é coordenada pelo serviço `AuthHandler`, executado uma única vez na inicialização da aplicação, que restaura o estado do usuário antes de qualquer rota ser renderizada. Para o gerenciamento de janelas modais, foi adotado um sistema de registro dinâmico de componentes, no qual cada modal é registrado por nome e pode ser exibido a partir de qualquer ponto da árvore de componentes, reduzindo o acoplamento. A interface oferece suporte a dois idiomas, português e inglês, com internacionalização implementada pela biblioteca `i18next`, que detecta o idioma do navegador; o conteúdo dinâmico é recuperado diretamente dos campos bilíngues armazenados no banco de dados.

4.3. Funcionalidades entregues

O conjunto de recursos implementados cobre os requisitos funcionais identificados. No gerenciamento de membros, professores podem convidar novos integrantes com informações pré-preenchidas, além de remover membros e editar funções, cargos e sala associada, atendendo ao objetivo específico O1. No inventário, o sistema permite o cadastro completo de equipamentos com identificador único, categoria, estado de conservação, localização por sala e seção, e responsável, com criação, edição e remoção restritas a professores e gerentes de inventário, atendendo ao objetivo específico O2. Projetos de pesquisa e áreas de atuação são cadastrados com títulos e descrições bilíngues e listas de membros participantes, e parcerias institucionais são exibidas na página pública com nome, logotipo e referência externa.

O sistema oferece ainda a personalização da identidade visual do laboratório, um sistema de convites com validade de sete dias e a exportação de relatórios consolidados em PDF e planilha (XLSX), o que concretiza o objetivo específico O5. A página pública apresenta membros, projetos, áreas de pesquisa, parcerias e informações de contato de forma responsiva e adaptada a diferentes tamanhos de tela, o que, aliado ao uso de *layout* responsivo nos módulos de gestão interna, concretiza os objetivos específicos O3 e O6. O sistema foi desenvolvido como *software* livre e de código aberto, com o código-fonte disponível publicamente, e a implantação em produção foi documentada em um guia que cobre a configuração do banco de dados PostgreSQL [PostgreSQL Global Development Group 2024], do servidor de aplicação uWSGI, do servidor web e do certificado de segurança.

5. Considerações finais

Este trabalho apresentou o desenvolvimento de um sistema web voltado ao gerenciamento compartilhado de laboratórios de pesquisa e desenvolvimento em ambientes acadêmicos. A motivação central partiu de um problema recorrente nesses espaços: a ausência de ferramentas integradas e acessíveis para organizar integrantes, equipamentos, projetos de pesquisa, parcerias e demais informações institucionais. O desenvolvimento foi conduzido com Django 6 no *backend*, React 19 e TypeScript no *frontend* e PostgreSQL como banco de dados, conforme planejado. A organização do *backend* em *apps* independentes mostrou-se adequada para separar responsabilidades e manter a coesão de cada módulo, e o *frontend*, estruturado como uma SPA, oferece uma experiência de navegação fluida e adaptada a diferentes dispositivos.

No que diz respeito às decisões técnicas, o armazenamento dos papéis de usuário como lista em formato JSON, em lugar do sistema de grupos nativo do Django, mostrou-se prático e de fácil manutenção. Da mesma forma, a adoção do método `export()` nos modelos, em lugar de serializadores do Django REST Framework, ofereceu controle direto sobre a estrutura das respostas da API sem introduzir camadas adicionais de abstração. O suporte bilíngue em português e inglês, implementado tanto nos campos do banco de dados quanto na interface, amplia o alcance do sistema.

Como trabalhos futuros, identificam-se algumas direções relevantes. A implantação em ambiente de produção com acesso real de usuários permitiria coletar evidências empíricas sobre a usabilidade e a aderência da solução às necessidades do contexto. A realização de testes formais com usuários finais, como professores e estudantes, possibilitaria identificar pontos de melhoria na interface e no fluxo de tarefas. Além disso, funcionalidades como notificações, reserva de equipamentos e integração com sistemas de autenticação institucionais, como a autenticação centralizada da UFSC [UFSC 2024], poderiam expandir a utilidade da plataforma. Em síntese, o sistema desenvolvido atinge os objetivos propostos, entregando uma aplicação funcional e documentada, com código-fonte disponível publicamente, de modo que qualquer laboratório interessado possa adaptar ou estender a solução conforme suas próprias necessidades.

Referências

Akeem Adekunle, A., Lukman Abolore, B., Mutiu, G., and Micheal Olalekan, A. (2024). Design and implementation of a web based laboratory management system (WBLMS):

- A case study of federal polytechnic, ile-oluji, ondo state. *Journal of Advanced Artificial Intelligence*, 1(2):15–20.
- Al-Salem, L. S. and Abu Samaha, A. (2007). Eliciting web application requirements – an industrial case study. *Journal of Systems and Software*, 80(3):294–313.
- Astral (2024). uv – an extremely fast python package manager. Acessado em: 2025-07-03. Disponível em: <https://docs.astral.sh/uv>.
- Auth0, Inc. (2015). JSON web tokens – introduction. Padrão definido pela RFC 7519. Acessado em: 2025-07-03. Disponível em: <https://jwt.io/introduction>.
- Chieu, T. C., Mohindra, A., Karve, A. A., and Segal, A. (2009). Dynamic scaling of web applications in a virtualized cloud computing environment. In *2009 IEEE International Conference on e-Business Engineering*, pages 281–286.
- Django Software Foundation (2025). Django documentation. Versão 6.0. Acessado em: 2025-07-03. Disponível em: <https://docs.djangoproject.com>.
- Fadilah, A. U., Widcaksono, T. A., Wonohardjo, E. P., and Yossy, E. H. (2021). Design of cadets administration system for nusantara cilacap maritime academy based on website. In *2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)*, volume 1, pages 142–147.
- Garousi, V., Mesbah, A., Betin-Can, A., and Mirshokraie, S. (2013). A systematic mapping study of web application testing. *Information and Software Technology*, 55(8):1374–1396.
- Manhas, J. (2017). Initial framework for website design and development. *International Journal of Information Technology*, 9(4):363–375.
- Molina-Ríos, J. and Pedreira-Souto, N. (2020). Comparison of development methodologies in web applications. *Information and Software Technology*, 119:106238.
- OWASP Foundation (2021). OWASP top 10:2021 – the ten most critical web application security risks. Acessado em: 2026-06-13. Disponível em: <https://owasp.org/Top10/>.
- PostgreSQL Global Development Group (2024). PostgreSQL documentation. Versão 17. Disponível em: <https://www.postgresql.org/docs/>.
- React Team (2024). React documentation. Versão 19.0. Disponível em: <https://react.dev>.
- Rosado, N., Metrolho, J. C., and Ribeiro, F. (2016). Web application for academic management – a case study. *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–7.
- Semastin, E., Azam, S., Shanmugam, B., Kannoorpatti, K., Jonokman, M., Narayana Samy, G., and Perumal, S. (2018). Preventive measures for cross site request forgery attacks on web-based applications. *International Journal of Engineering and Technology (UAE)*, 7:130–134.
- Tanenbaum, A. S. and Van Steen, M. (2007). *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2 edition.

- Timoteo, M. O., Brochado, A. C. B., Costa-Silva, D., Granjeiro, J. M., Olej, B., and Alves, G. G. (2020). The impact of online management systems: a qualitative assessment of staff perception at a clinical research laboratory. *Research, Society and Development*, 9(10).
- UFSC (2024). Autenticação centralizada ufsc. Acessado em: 2026-06-10. Disponível em: <https://servicosti.sistemas.ufsc.br/publico/detalhes.xhtml?servico=355>.
- Unbit (2014). uwsgi application server. Versão 2.0. Acessado em: 2025-07-03. Disponível em: <https://uwsgi-docs.readthedocs.io/>.
- Vincent, W. (2022). *Django for Professionals*. Independently Published.
- VoidZero, Inc. (2024). Vite documentation. Versão 6.0. Disponível em: <https://vite.dev/guide/>.
- Vujic, I. (2022). Hashids – generate short unique ids from integers. Versão 1.3.1. Acessado em: 2025-07-03. Disponível em: <https://hashids.org>.
- Wu, L., Lu, W., and Chen, C. (2024). Strengths and weaknesses of client-server and peer-to-peer network models in construction projects. *International Journal of Construction Management*, 24(12):1349–1363.

APÊNDICE B – INSTRUÇÕES PARA EXECUÇÃO LOCAL DO PROJETO

O código-fonte do projeto está disponível nos seguintes repositórios públicos:

- <https://github.com/gustavogds/lab-manager>
- <https://codigos.ufsc.br/gustavo.gs/lab-manager>

As instruções a seguir descrevem como executar o sistema localmente em um ambiente Ubuntu/Linux, assumindo que o repositório já foi clonado. É necessário que o Python 3, o Node.js, o uv (Astral, 2024) e o PostgreSQL estejam instalados no sistema.

B.1 CONFIGURAÇÃO DO AMBIENTE PYTHON

O gerenciamento de dependências Python é feito com o uv. Para instalar as dependências do projeto, execute:

```
uv sync --group dev
```

B.2 CONFIGURAÇÃO DO FRONTEND

Para instalar as dependências do *frontend* e gerar o pacote estático de produção, execute:

```
cd client/  
npm install  
npm run build
```

B.3 CONFIGURAÇÃO DO BANCO DE DADOS

Certifique-se de que o PostgreSQL está instalado e em execução. Para criar o banco de dados e o usuário, execute:

```
sudo -u postgres psql  
CREATE DATABASE nome_do_banco;  
CREATE USER nome_do_usuario WITH PASSWORD 'senha_do_usuario';  
GRANT ALL PRIVILEGES ON DATABASE nome_do_banco TO nome_do_usuario;
```

Em seguida, crie o arquivo `.env` dentro do diretório `server/` com as variáveis de ambiente:

```
DB_NAME=<Nome do Banco de Dados>  
DB_USER=<Usuario do Banco>
```

```
DB_PASSWORD=<Senha do Banco>  
DB_PORT=5432  
SITE_URL=http://localhost:8000
```

B.4 INICIALIZAÇÃO DO BACKEND

Execute as migrações e colete os arquivos estáticos:

```
uv run python server/manage.py migrate  
uv run python server/manage.py collectstatic --noinput
```

Para criar um usuário administrador inicial:

```
uv run python server/manage.py createsuperuser
```

B.5 EXECUÇÃO DO SERVIDOR

Inicie o servidor com o uWSGI, que sobe o Django e o servidor de desenvolvimento do Vite em paralelo:

```
uwsgi extras/uwsgi.ini
```

A aplicação estará disponível em `http://localhost:8000`.

APÊNDICE C – INSTRUÇÕES PARA IMPLANTAÇÃO EM AMBIENTE DE PRODUÇÃO

Este apêndice descreve o procedimento para implantação do Lab Manager em um servidor Linux com acesso público à internet, utilizando uWSGI (Unbit, 2014) como servidor de aplicação, Nginx (nginx, 2026) como servidor *web* reverso e PostgreSQL como banco de dados. O guia considera um servidor Ubuntu 22.04 ou Debian 12 com acesso *SSH* e um domínio já apontando para o IP do servidor. O guia completo, incluindo procedimentos de atualização (*re-deploy*), *checklist* de segurança e resolução de problemas, está disponível no repositório do projeto.

C.1 PRÉ-REQUISITOS DO SERVIDOR

Os requisitos mínimos de infraestrutura são: 1 GB de RAM (recomendado 2 GB ou mais), 10 GB de espaço em disco e as portas TCP 22 (*SSH*), 80 (*HTTP*) e 443 (*HTTPS*) abertas tanto no firewall do sistema operacional quanto no painel do provedor de nuvem. Provedores como AWS, DigitalOcean e GCP possuem firewalls próprios no nível de rede que operam de forma independente ao firewall do servidor e devem ser configurados previamente.

C.2 INSTALAÇÃO DE DEPENDÊNCIAS DO SISTEMA

Conecte ao servidor via *SSH* e execute a atualização do sistema e a instalação das dependências essenciais:

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y build-essential postgresql postgresql-contrib \
    libpq-dev nginx curl git certbot python3-certbot-nginx
```

O projeto utiliza Python 3.12, cuja versão é gerenciada pelo uv (Astral, 2024). O Node.js é necessário apenas para gerar o pacote estático do *frontend*:

```
# Instalar Node.js 20+
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install -y nodejs

# Instalar o uv
curl -LsSf https://astral.sh/uv/install.sh | sh
source ~/.bashrc
```

C.3 CONFIGURAÇÃO DO POSTGRESQL

Acesse o shell do PostgreSQL e crie o banco de dados e o usuário da aplicação:

```
sudo -u postgres psql

CREATE DATABASE lab_manager;
CREATE USER lab_manager_user WITH PASSWORD 'senha_forte';
ALTER ROLE lab_manager_user SET client_encoding TO 'utf8';
ALTER ROLE lab_manager_user SET default_transaction_isolation
    TO 'read committed';
ALTER ROLE lab_manager_user SET timezone TO 'UTC';
GRANT ALL PRIVILEGES ON DATABASE lab_manager TO lab_manager_user;
ALTER DATABASE lab_manager OWNER TO lab_manager_user;
```

↵

C.4 OBTENÇÃO DO CÓDIGO-FONTE

Clone a *branch* master (branch de produção) no diretório de instalação:

```
sudo mkdir -p /opt/lab-manager
sudo chown ubuntu:ubuntu /opt/lab-manager # adapte para o seu usuário
git clone -b master \
    https://github.com/gustavogds/lab-manager.git \
    /opt/lab-manager/app
cd /opt/lab-manager/app
```

C.5 CONFIGURAÇÃO DO AMBIENTE PYTHON

O uv instala o Python 3.12 automaticamente e cria o ambiente virtual isolado:

```
cd /opt/lab-manager/app
uv python install 3.12
uv venv --python 3.12
source .venv/bin/activate
uv sync --frozen
deactivate
```

C.6 BUILD DO FRONTEND

Instale as dependências do Node.js e gere o pacote de produção do *frontend*:

```
cd /opt/lab-manager/app/client
npm install
NODE_ENV=production npm run build
```

O build bem-sucedido gera o arquivo `server/staticfiles/client/.vite/manifest.json`, que é exigido pelo Django para carregar o *frontend* em produção.

C.7 ARQUIVO DE CONFIGURAÇÃO DE AMBIENTE

Crie o arquivo `server/.env` com as variáveis de ambiente da instância de produção:

```
DB_NAME=lab_manager
DB_USER=lab_manager_user
DB_PASSWORD=senha_forte
DB_HOST=localhost
DB_PORT=5432
SITE_URL=https://labmanager.seudominio.com
SECRET_KEY=chave_gerada
JWT_SECRET=valor_gerado
GUIDS_SECRET=valor_gerado
EMAIL_BACKEND=django.core.mail.backends.smtp.EmailBackend
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USE_TLS=True
EMAIL_HOST_USER=seuemail@gmail.com
EMAIL_HOST_PASSWORD=app_password
DEFAULT_FROM_EMAIL=seuemail@gmail.com
```

As chaves criptográficas (`SECRET_KEY`, `JWT_SECRET` e `GUIDS_SECRET`) devem ser geradas com valores aleatórios únicos para cada implantação. O arquivo deve ter permissão de leitura restrita ao proprietário:

```
chmod 600 /opt/lab-manager/app/server/.env
```

C.8 VERIFICAÇÃO DAS CONFIGURAÇÕES DE PRODUÇÃO

A *branch* `master` já contém as configurações de produção no `settings.py`, incluindo `DEBUG = False`, suporte ao `WhiteNoise` para arquivos estáticos, redirecionamento `HTTPS` e `HSTS`. O único ajuste necessário é incluir o domínio da instância em `ALLOWED_HOSTS`:

```
ALLOWED_HOSTS = ["labmanager.seudominio.com", "127.0.0.1", "localhost"]
```

C.9 PREPARAÇÃO DO DJANGO

Crie os diretórios de armazenamento em tempo de execução e execute as rotinas de inicialização do Django:

```
mkdir -p /opt/lab-manager/app/server/storage/{files,private/objects,
    private/thumbs,upload-tmp,logs}

cd /opt/lab-manager/app
source .venv/bin/activate
python server/manage.py migrate
python server/manage.py collectstatic --noinput
python server/manage.py createsuperuser
deactivate
```

C.10 CONFIGURAÇÃO DO UWSGI PARA PRODUÇÃO

Crie o arquivo `extras/uwsgi-production.ini` com a configuração de produção do servidor de aplicação. Os parâmetros principais estão descritos a seguir; o número de processos deve ser ajustado conforme os recursos do servidor (regra geral: $2 \times \text{núcleos} + 1$):

```
[uwsgi]
master = true
enable-threads = true
chdir = /opt/lab-manager/app/server
home = /opt/lab-manager/app/.venv
module = config.wsgi:application
env = DJANGO_SETTINGS_MODULE=config.settings
env = PYTHON_ENV=production
socket = /opt/lab-manager/app/server/storage/lab-manager.sock
chmod-socket = 660
processes = 5
threads = 2
max-requests = 5000
harakiri = 120
die-on-term = true
vacuum = true
offload-threads = 4
buffer-size = 65535
pidfile = /opt/lab-manager/app/server/storage/uwsgi.pid
```

C.11 CONFIGURAÇÃO DO NGINX

Crie o arquivo `/etc/nginx/sites-available/lab-manager` para configurar o servidor *web* reverso. Nesta etapa, o Nginx é configurado apenas para HTTP (porta 80); o HTTPS será adicionado na etapa seguinte, após a obtenção do certificado SSL:

```
server {
    listen 80;
    server_name labmanager.seudominio.com;
    client_max_body_size 50M;

    location /static/ {
        alias /opt/lab-manager/app/server/staticfiles/;
        expires 30d;
        add_header Cache-Control "public, immutable";
    }
    location /media/ {
        alias /opt/lab-manager/app/server/storage/files/;
    }
    location /media/object/ {
        alias /opt/lab-manager/app/server/storage/private/objects/;
        internal;
    }
    location /media/thumb/ {
        alias /opt/lab-manager/app/server/storage/private/thumbs/;
        internal;
    }
    location / {
        include uwsgi_params;
        uwsgi_pass unix:/opt/lab-manager/app/server/storage/lab-manager.sock;
    }
    location /.well-known/acme-challenge/ {
        root /var/www/html;
    }
}
```

Ative o *site* e reinicie o Nginx:

```
sudo ln -s /etc/nginx/sites-available/lab-manager \
    /etc/nginx/sites-enabled/
sudo rm -f /etc/nginx/sites-enabled/default
sudo nginx -t && sudo systemctl start nginx
sudo systemctl enable nginx
```

C.12 SERVIÇO SYSTEMD PARA O UWSGI

Crie o arquivo de unidade `/etc/systemd/system/lab-manager.service` para que o sistema operacional gerencie o ciclo de vida do uWSGI:

```
[Unit]
Description=Lab Manager uWSGI Application
After=network.target postgresql.service
Requires=postgresql.service

[Service]
Type=notify
User=ubuntu
Group=ubuntu
WorkingDirectory=/opt/lab-manager/app/server
ExecStart=/opt/lab-manager/app/.venv/bin/uwsgi \
    --ini /opt/lab-manager/app/extras/uwsgi-production.ini
ExecReload=/bin/kill -HUP $MAINPID
ExecStop=/bin/kill -INT $MAINPID
Restart=on-failure
RestartSec=5
KillSignal=SIGQUIT
TimeoutStopSec=30
PrivateTmp=true
NoNewPrivileges=true

[Install]
WantedBy=multi-user.target
```

Ative e inicie o serviço:

```
sudo systemctl daemon-reload
sudo systemctl start lab-manager
sudo systemctl enable lab-manager
sudo systemctl status lab-manager
```

C.13 CERTIFICADO SSL COM LET'S ENCRYPT

Com o uWSGI em execução e o domínio apontando corretamente para o servidor, obtenha o certificado TLS/SSL de forma automatizada pelo Certbot (Electronic Frontier Foundation, 2026):

```
sudo certbot --nginx -d labmanager.seudominio.com
```

O Certbot valida o controle do domínio via HTTP, obtém o certificado, modifica automaticamente a configuração do Nginx para HTTPS e configura o redirecionamento de HTTP para HTTPS. A renovação automática é habilitada pelo Certbot como um *timer* do systemd, dispensando intervenção manual. Para verificar:

```
sudo certbot renew --dry-run
```

C.14 CONFIGURAÇÃO DO FIREWALL

Configure o firewall UFW para liberar apenas as portas necessárias. A permissão de SSH deve ser adicionada antes de ativar o firewall para evitar bloqueio de acesso ao servidor:

```
sudo ufw allow OpenSSH
sudo ufw allow 'Nginx Full'
sudo ufw enable
sudo ufw status verbose
```

C.15 PERMISSÕES DE ARQUIVOS

Garanta que o usuário do sistema seja proprietário de todos os arquivos da aplicação e que o diretório de armazenamento tenha permissão de escrita:

```
sudo chown -R ubuntu:ubuntu /opt/lab-manager/
chmod -R 755 /opt/lab-manager/app/server/storage/
chmod 600 /opt/lab-manager/app/server/.env
```

Concluídas estas etapas, a aplicação estará acessível via HTTPS no domínio configurado. Os procedimentos complementares, como atualização da aplicação (*re-deploy*), *checklist* de segurança e resolução de problemas comuns, estão documentados no guia completo¹ disponível no repositório do projeto.

¹<https://github.com/gustavogds/lab-manager/blob/develop/DEPLOY.md>