UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Murilo Medeiros do Valle

Modernização tecnológica de Front-End: uma proposta de interface de interação com usuário para o Classroom eXperience

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Murilo Medeiros do Valle

Modernização tecnológica de Front-End: uma proposta de interface de interação com usuário para o Classroom eXperience

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: **Prof. Dr. Rafael Dias Araújo**

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2025

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

V181 2025

Valle, Murilo Medeiros do, 1999-
Modernização tecnológica de Front-End: uma proposta de interface de interação com usuário para o Classroom eXperience [recurso eletrônico] / Murilo Medeiros do Valle. - 2025.

Orientador: Rafael Dias Araújo.
Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Uberlândia, Graduação em Sistemas de Informação.
Modo de acesso: Internet.
Inclui bibliografia.
Inclui ilustrações.

1. Tecnologia da informação - Administração. I. Araújo, Rafael Dias,1986-, (Orient.). II. Universidade Federal de Uberlândia. Graduação em Sistemas de Informação. III. Título.

CDU: 658:681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:
Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

Murilo Medeiros do Valle

Modernização tecnológica de Front-End: uma proposta de interface de interação com usuário para o Classroom eXperience

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 12 de maio de 2025:

Prof. Dr. Rafael Dias Araújo
(FACOM/UFU)
Orientador

Prof. Dr. Fabiano Azevedo Dorça
(FACOM/UFU)

Prof. Dra. Maria Adriana Vidigal de Lima (FACOM/UFU)

Uberlândia, Brasil
2025

Resumo

A evolução tecnológica constante no desenvolvimento de software exige que sistemas antigos sejam modernizados para atender às demandas atuais de usabilidade, escalabilidade e manutenção. Este trabalho propõe a modernização tecnológica do *Front-End* da plataforma educacional *Classroom eXperience (CX)*, visando melhorar sua estrutura, desempenho e experiência do usuário. A partir da análise do sistema original e de seus requisitos, foram definidas tecnologias modernas como Angular e TypeScript, além da adoção da *Clean Architecture* para organização do código. O processo incluiu a reestruturação de páginas, implementação de autenticação baseada em *tokens*, internacionalização e melhorias na documentação e integração contínua. Os resultados obtidos indicam avanços na performance, organização e escalabilidade do sistema, possibilitando uma base sólida para futuras evoluções da plataforma.

Palavras-chave: *Front-End*; Modernização de Software; Angular; *Clean Architecture*; *Classroom eXperience*; Experiência do Usuário; Migração de Sistemas; Desenvolvimento *Web*.

Lista de ilustrações

Figura 1 – Diretório `src` . . . 38
Figura 2 – Diretório `app` . . . 39
Figura 3 – Modelo aplicado da arquitetura limpa no sistema . . . 40
Figura 4 – Diretório `register` . . . 41
Figura 5 – Arquivo `access-restrictions.model.ts` . . . 43
Figura 6 – Arquivo `access-restriction-entity.png` . . . 44
Figura 7 – Arquivo `access-restriction-mapper.png` . . . 45
Figura 8 – Classe `StorageBase` . . . 46
Figura 9 – Classe `SessionStorageService` (à esquerda) e classe `LocalStorageService` (à direita) . . . 47
Figura 10 – Classe `AuthService` . . . 47
Figura 11 – Classe `TranslationService` . . . 47
Figura 12 – Classe `AuthService` . . . 48
Figura 13 – *Auth Interceptor* . . . 50
Figura 14 – Classe `TranslationService` . . . 51
Figura 15 – *Translation Interceptor* . . . 52
Figura 16 – Diretório raiz do projeto, destacando o diretório `documentation` e o arquivo `README` . . . 53
Figura 17 – `README.md` . . . 54
Figura 18 – `ENVIRONMENT.md` . . . 55
Figura 19 – Arquivo `.gitlab-ci.yml` . . . 57
Figura 20 – Exemplo de *pipeline* executada com sucesso . . . 58
Figura 21 – Exemplo de *pipeline* com falha na etapa de testes . . . 58
Figura 22 – Exemplo de *pipeline* com falha na etapa de `build` . . . 59
Figura 23 – Página de *Login* [*OLD*] . . . 61
Figura 24 – Página de *Login* [*NEW*] . . . 61
Figura 25 – Página de Registro [*OLD*] . . . 62
Figura 26 – Página de Registro [*NEW*] . . . 62
Figura 27 – Página de Disciplinas [*OLD*] . . . 63
Figura 28 – Página de Disciplinas [*NEW*] . . . 63
Figura 29 – Página da Disciplina [*OLD*] . . . 64
Figura 30 – Página de Desempenho na Disciplina [*OLD*] . . . 64
Figura 31 – Página da Disciplina [*NEW*] . . . 65
Figura 32 – Página das Aulas da Disciplina [*NEW*] . . . 65
Figura 33 – Página de Dados Pessoais [*OLD*] . . . 66
Figura 34 – Página de Dados Pessoais [*NEW*] . . . 66

Figura 35 – Página de Contexto de Acesso [*OLD*] 67
Figura 36 – Página de Contexto de Acesso [*NEW*] 67
Figura 37 – Página de Restrições de Acesso [*OLD*] 68
Figura 38 – Página de Restrições de Acesso [*NEW*] 68
Figura 39 – Página de Criação de Restrição de Acesso [*OLD*] 69
Figura 40 – Página de Criação de Restrição de Acesso [*NEW*] 69
Figura 41 – Página de Preferências [*OLD*] 70
Figura 42 – Página de Preferências [*NEW*] 70
Figura 43 – Página de ILS [*OLD*] 71
Figura 44 – Página de ILS [*NEW*] 71
Figura 45 – Página de Conquistas [*OLD*] 72
Figura 46 – Página de Conquistas [*NEW*] 72
Figura 47 – Página de *Login Mobile* 73
Figura 48 – Página de Registro *Mobile* 73
Figura 49 – Página de Disciplinas *Mobile* 74
Figura 50 – Página da Disciplina *Mobile* 74
Figura 51 – Página de Dados Pessoais *Mobile* 75
Figura 52 – Página de Restrições de Acesso *Mobile* 75
Figura 53 – Lighthouse — Página de *login* antiga (*Mobile*) 76
Figura 54 – Lighthouse — Página de *login* nova (*Mobile*) 76
Figura 55 – Lighthouse — Página de *login* antiga (*Desktop*) 77
Figura 56 – Lighthouse — Página de *login* nova (*Desktop*) 77
Figura 57 – Lighthouse — Página de disciplinas nova (*Mobile*) 78
Figura 58 – Lighthouse — Página de disciplinas nova (*Desktop*) 78

Lista de Abreviaturas e Siglas

API Application Programming Interface

C&A Captura e Acesso

CD Continuous Delivery

CI Continuous Integration

CI/CD Continuous Integration and Continuous Delivery

CIA Confidentiality, Integrity, Availability

CLI Command-Line Interface

CSS Cascading Style Sheets

CX Classroom eXperience

DCU Design Centrado no Usuário

DevOps Development and Operations

DevSecOps Development, Security and Operations

GAIA Guia de Acessibilidade de Interfaces para Autismo

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IA Inteligência Artificial

ILS Index of Learning Styles

JS JavaScript

JSON JavaScript Object Notation

JWT JSON Web Token

MFA Multi-Factor Authentication

ML Machine Learning

MR Merge Request

NoSQL Not Only SQL

POLP Principle of Least Privilege

PWA Progressive Web App

REST Representational State Transfer

RF Requisitos Funcionais

RNF Requisitos Não-Funcionais

SASS Syntactically Awesome Style Sheets

SCSS Sassy Cascading Style Sheets

SEO Search Engine Optimization

SIGA Sistema de Gestão Acadêmica

SOLID Single Responsibility, Open–Closed, Liskov Substitution, Interface Segregation, Dependency Inversion

TS TypeScript

UI User Interface

UX User Experience

UX/UI User Experience / User Interface

VCS Version Control Systems

WWW World Wide Web

YAML Yet Another Markup Language

Sumário

1 INTRODUÇÃO 12
1.1 Objetivos 14
1.2 Organização do texto 14

2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 Fundamentos do Desenvolvimento *Front-End* 16
2.1.1 Histórico 16
2.1.2 Avanços recentes da área 16
2.1.3 Exemplos 17
2.2 Arquitetura para Front-End 18
2.2.1 Histórico 18
2.2.2 Avanços recentes da área 18
2.2.3 Exemplos 19
2.3 Migração de Sistemas Legados 19
2.3.1 Análise de projetos 19
2.3.1.1 Principais Definições 19
2.3.1.2 Histórico 20
2.3.1.3 Avanços recentes da área 20
2.3.1.4 Exemplos 20
2.3.2 Estratégias para migração de sistemas 20
2.3.2.1 Principais Definições 20
2.3.2.2 Histórico 21
2.3.2.3 Avanços recentes da área 21
2.3.2.4 Exemplos 21
2.4 Experiência do Usuário (User Experience (UX)) 22
2.4.1 Principais Definições 22
2.4.2 Histórico 22
2.4.3 Avanços recentes da área 22
2.4.4 Exemplos 22
2.5 Segurança em Sistemas de Informação 23
2.5.1 Principais Definições 23
2.5.2 Histórico 23
2.5.3 Avanços recentes da área 23
2.5.4 Exemplos 24
2.6 Versionamento de Código 24

2.6.1 Principais Definições . . . 24
2.6.2 Histórico . . . 25
2.6.3 Avanços Recentes na Área . . . 25
2.6.4 Exemplos . . . 25
2.7 Documentação e Transferência de Conhecimento . . . 26
2.7.1 Principais Definições . . . 26
2.7.2 Histórico . . . 26
2.7.3 Avanços recentes da área . . . 26
2.7.4 Exemplos . . . 27

3 TRABALHOS RELACIONADOS . . . 28

4 DESENVOLVIMENTO . . . 30
4.1 Planejamento . . . 30
4.1.1 Análise do Estado Atual do Sistema . . . 30
4.1.2 Requisitos do Sistema . . . 31
4.1.2.1 Requisitos Funcionais . . . 31
4.1.2.2 Requisitos Não-Funcionais . . . 33
4.1.3 Modelagem de dados e contratos da API . . . 33
4.1.3.1 Exemplo de Contrato 1 - *Login* de Usuário . . . 34
4.1.3.2 Exemplo de Contrato 2 - Buscar Disciplinas por Curso . . . 34
4.2 Tecnologias Utilizadas . . . 35
4.2.1 Angular . . . 35
4.2.2 TypeScript . . . 36
4.2.3 Angular Material . . . 36
4.2.4 Interface de Programação de Aplicação (Application Programming Interface (API)) . . . 37
4.3 Implementação do Sistema . . . 37
4.3.1 Arquitetura Limpa . . . 37
4.3.1.1 *Model*: `AccessRestrictionModel` . . . 42
4.3.1.2 *Entity*: `AccessRestrictionEntity` . . . 42
4.3.1.3 *Mapper*: `AccessRestrictionMapper` . . . 44
4.3.2 Armazenamento Local . . . 44
4.3.3 Autenticação do Usuário . . . 47
4.3.4 Internacionalização . . . 49
4.3.5 Documentação do Sistema . . . 52
4.3.6 Controle de Versão e Repositório de Código . . . 54
4.3.7 Entrega e Integração Contínua (Continuous Integration and Continuous Delivery (CI/CD)) . . . 56

5 RESULTADOS 60
5.1 Visuais e de Usabilidade 60
5.2 Resultados de Desempenho e Qualidade Técnica 75

6 CONCLUSÃO 79

REFERÊNCIAS 80

1 Introdução

Em um mundo moderno, onde a computação está presente em nossas vidas de forma onipresente, não se pode deixar de notar que os sistemas educacionais também evoluíram, se enraizaram e aproveitaram muitos dos avanços que a tecnologia trouxe para essa nova era. Deve-se levar em consideração, também, que, durante os últimos anos dessa evolução, a quantidade de dados trafegados pela Internet aumentou significativamente (Richard Santa, 2018), demandando a utilização de diferentes tipos de mídias para atender a distintos perfis de público que buscam experiências personalizadas para suas necessidades específicas. Como destacado por (SODRÉ et al., 2021), a tecnologia na educação não apenas facilita o acesso à informação, mas também promove a inclusão digital, especialmente em um contexto no qual a pandemia de COVID-19 acelerou a necessidade de integração de ferramentas tecnológicas no ensino. No entanto, a exclusão digital ainda é um desafio relevante, exigindo políticas públicas que garantam o acesso equitativo a esses recursos para todos os estudantes, inclusive aqueles com deficiências (SODRÉ et al., 2021).

Nesse cenário, a modernização de sistemas legados tem-se tornado uma necessidade constante diante da rápida evolução tecnológica. Ambientes computacionais antigos, embora funcionais, frequentemente apresentam limitações quanto à escalabilidade, à segurança e à experiência do usuário, dificultando a integração com novas ferramentas e práticas modernas, como o uso de arquiteturas desacopladas, testes automatizados e metodologias de desenvolvimento ágil. A atualização desses sistemas, especialmente no contexto *Web*, é essencial para garantir maior flexibilidade, facilitar a manutenção e proporcionar uma base mais robusta para evoluções futuras. Assim, a modernização se consolida não apenas como um aprimoramento visual ou funcional, mas como um fator estratégico para a sustentabilidade e longevidade de soluções digitais (Tasmiha Khan, 2023).

Com o objetivo de contribuir para os avanços no ensino mediado por tecnologia e promover uma educação de maior qualidade, foi desenvolvida a plataforma denominada *Classroom eXperience (CX)*, proposta por (CATTELAN et al., 2024). O CX é uma plataforma educacional voltada ao suporte do processo de ensino-aprendizagem em ambientes presenciais, remotos e híbridos. Seu propósito é oferecer uma experiência de aprendizagem mais rica e personalizada por meio da captura, organização e disponibilização de conteúdos multimídia estruturados.

A plataforma se baseia no conceito de Captura e Acesso (C&A) (Captura e Acesso), permitindo que docentes consolidem os materiais preparados previamente, realizem a gra-

vação da aula ao vivo — incluindo áudio, vídeo e interações em slides ou lousa digital — e executem o pós-processamento dessas mídias. Em seguida, os conteúdos são disponibilizados em um ambiente organizado e acessível para os estudantes, que podem revisitar o material em diferentes momentos, complementá-lo com anotações pessoais e organizar seu próprio processo de estudo.

Além disso, o CX oferece recursos voltados à usabilidade e à personalização da experiência do usuário, como ferramentas de anotação, inserção de marcadores temporais (*bookmarks*) e funcionalidades colaborativas. Conforme demonstrado nos trabalhos anteriores sobre a plataforma, a interação contínua entre professores e alunos resultou em melhorias significativas na dinâmica de aprendizagem. Um exemplo notável são os *bookmarks*, amplamente aceitos pelos estudantes e enriquecidos com funcionalidades como preenchimento automático, sugestão de marcadores e visualização de anotações previamente criadas (ARAÚJO et al., 2016).

Dessa forma, o CX não apenas registra e organiza o conteúdo didático, mas também promove um ambiente de estudo centrado no aluno, favorecendo a autonomia, a personalização e o engajamento contínuo no processo educacional.

Além disso, a plataforma CX incorpora o conceito de Estilos de Aprendizagem, fundamentado no modelo proposto por Felder e Silverman (1988), operacionalizado por meio do Índice de Estilos de Aprendizagem (ILS). Esse instrumento avalia as preferências dos estudantes em quatro dimensões: ativo/reflexivo, sensorial/intuitivo, visual/verbal e sequencial/global. Ao identificar essas preferências, o CX permite personalizar a apresentação dos conteúdos educacionais, adaptando-os às necessidades individuais dos alunos. Essa abordagem visa otimizar o processo de ensino-aprendizagem, promovendo maior engajamento e eficácia na assimilação dos conteúdos. A utilização do ILS na plataforma CX está alinhada com estudos que demonstram a relevância dos estilos de aprendizagem na melhoria do desempenho acadêmico e na personalização do ensino (FELDER; SILVERMAN, 1988).

Apesar de alcançar seu propósito inicial de fomentar a colaboração dinâmica entre os usuários, a plataforma tornou-se tecnologicamente obsoleta com o passar do tempo, resultando em problemas como lentidão, travamentos e falhas recorrentes. Além disso, sua estrutura monolítica, na qual o *Front-End* e o *Back-End* coexistem em um único repositório de código, comprometeu a modularidade do sistema. Essa arquitetura dificultava a manutenção, testabilidade e a identificação de falhas, representando um desafio para desenvolvedores que desejassem atuar em sua evolução.

É importante destacar que o processo de redesenho da interface da plataforma, incluindo a criação dos protótipos no Figma, os novos estilos visuais e a aplicação de práticas relacionadas a UX/UI, foi desenvolvido por outro discente durante seu projeto de Iniciação Científica. Essas atividades foram fundamentais para a modernização do sistema,

mas encontram-se fora do escopo do presente trabalho, que tem como foco principal a reconstrução do *Front-End* do sistema a partir dessas definições visuais já consolidadas.

1.1 Objetivos

O objetivo geral deste trabalho é modernizar e aprimorar a plataforma *Classroom eXperience*, com ênfase na reconstrução de sua interface *Front-End*, de modo a adequá-la às boas práticas de engenharia de software e às exigências técnicas atuais. Busca-se, com isso, promover uma maior escalabilidade, manutenibilidade e eficiência no desenvolvimento da aplicação *web*, preservando suas funcionalidades essenciais e possibilitando sua futura expansão.

Para alcançar tal propósito, foram definidos os seguintes objetivos específicos:

- Analisar o estado atual do sistema legado, identificando suas limitações técnicas e estruturais;
- Mapear e documentar os requisitos funcionais e não funcionais da plataforma;
- Redefinir a arquitetura da aplicação com base na abordagem *Clean Architecture*, promovendo modularidade e separação de responsabilidades;
- Implementar a nova versão do *Front-End* utilizando tecnologias modernas e compatíveis com os padrões atuais da indústria;
- Integrar boas práticas de programação e princípios de qualidade de software, como o *Single Responsibility, Open–Closed, Liskov Substitution, Interface Segregation, Dependency Inversion (SOLID)* e o código limpo;
- Estabelecer um fluxo de versionamento de código eficiente, com suporte à integração e entrega contínua (CI/CD);
- Documentar os processos, decisões técnicas e estrutura do projeto, favorecendo a manutenção e evolução do sistema por futuros desenvolvedores.

1.2 Organização do texto

O presente trabalho está estruturado em seis capítulos, organizados de forma lógica para que o leitor possa percorrer da fundamentação teórica à prática e conclusão do mesmo. No Capítulo 2 (Fundamentação Teórica), são abordados alguns conceitos importantes para a compreensão do projeto, tais como desenvolvimento *Front-End*, arquitetura de software, experiência do usuário (UX), migração de sistemas legados e segurança em

sistemas de informação. Essa seção sustenta teoricamente as decisões técnicas feitas para o desenvolvimento do mesmo.

No Capítulo 3 (Trabalhos Relacionados), são apresentados estudos e experiências acadêmicas e corporativas que tratam da modernização de sistemas *web*, especialmente no contexto de interfaces *Front-End*. A revisão contempla a abordagem de metodologias de migração, reestruturação arquitetural e melhorias na experiência do usuário, contribuindo com fundamentos teóricos e práticos que embasam as decisões deste trabalho.

O Capítulo 4 (Desenvolvimento) apresenta todas as etapas práticas e conceituais do projeto. Inicialmente, é feita uma análise do estado atual do sistema, seguida da definição de seus requisitos funcionais e não funcionais, bem como da modelagem de dados e contratos de API. Em seguida, são detalhadas as tecnologias adotadas (Angular, TypeScript, Material Design e APIs), com a devida justificativa para suas escolhas, considerando critérios como escalabilidade, manutenibilidade e aderência aos padrões de mercado. Por fim, descreve-se a implementação técnica do sistema, abordando tópicos como arquitetura limpa, armazenamento local, autenticação do usuário, internacionalização, documentação, versionamento de código e práticas de entrega e integração contínua (CI/CD).

O Capítulo 5 (Resultados) apresenta os principais resultados obtidos com o desenvolvimento do projeto, destacando as melhorias visuais e funcionais alcançadas. Essa análise é feita por meio de comparações entre a interface anterior e a nova versão da plataforma, evidenciando os avanços em termos de usabilidade, organização da informação e adequação às necessidades dos usuários.

O Capítulo 6 (Conclusão) sintetiza o que foi discutido no texto, retoma os principais aprendizados do processo de desenvolvimento e propõe orientações para o futuro da plataforma CX. Complementarmente, é feita uma reflexão sobre os desafios encontrados durante a modernização do sistema e sobre o cenário atual do desenvolvimento *Front-End*, destacando suas tendências e complexidades.

2 Fundamentação Teórica

2.1 Fundamentos do Desenvolvimento *Front-End*

Os termos *Front-End* e *Back-End* são amplamente utilizados no contexto do desenvolvimento de software. De forma geral, o *Front-End* refere-se à camada de apresentação de um sistema, ou seja, à interface visual com a qual o usuário interage diretamente. Cabe ao desenvolvedor dessa camada a criação de páginas, telas, botões, formulários, componentes gráficos e demais elementos que compõem a experiência do usuário, incluindo aspectos relacionados à interatividade e acessibilidade.

Por sua vez, o *Back-End* corresponde à parte interna do sistema, sendo responsável pelo processamento de dados, regras de negócio e comunicação com bancos de dados ou outros serviços. O *Front-End* geralmente se comunica com o *Back-End* para executar funcionalidades dinâmicas, como registrar novos usuários, recuperar informações armazenadas ou remover dados previamente cadastrados. Essas ações são processadas pelo servidor e seus resultados são enviados de volta à interface do usuário.

2.1.1 Histórico

Com a popularização da Internet, surgiu a necessidade de estruturar e apresentar o conteúdo disponibilizado por meio da *World Wide Web (WWW)*. Para atender a essa demanda, foram desenvolvidas ferramentas específicas para a criação de interfaces a partir de código-fonte. Nesse contexto, destacam-se as linguagens *Hypertext Markup Language (HTML)*, *Cascading Style Sheets (CSS)* e *JavaScript (JS)*, que inicialmente permitiam apenas a construção de páginas estáticas. Atualmente, esses recursos constituem a base fundamental para o desenvolvimento de aplicações *web* dinâmicas, personalizáveis e amplamente acessíveis em escala global (Rafaella Ballerini, Time Alura, 2023).

2.1.2 Avanços recentes da área

Desde seu surgimento, a área de desenvolvimento *Front-End* tem evoluído consideravelmente, demonstrando sua crescente relevância em diversos setores, como empresas, instituições públicas e projetos individuais. Esse avanço contínuo possibilitou o surgimento de novas tecnologias, *frameworks* e paradigmas de desenvolvimento que transformaram a maneira como aplicações *web* são projetadas e entregues aos usuários. Dentre os principais avanços, destacam-se:

- *Frameworks* e bibliotecas *JavaScript*: a crescente demanda por interfaces mais mo-

dulares, reativas e escaláveis impulsionou o desenvolvimento de bibliotecas e *frameworks* como *React* (react.dev), *Angular* (angular.dev) e *Vue.js* (vuejs.org), que tornaram-se fundamentais para a criação de aplicações modernas e eficientes.

- *Progressive Web App (PWA)*: os PWAs combinam a flexibilidade das aplicações *web* com características típicas de aplicativos nativos, como funcionamento offline, notificações e instalação direta no dispositivo, oferecendo uma experiência mais completa e fluida ao usuário.

- *Design Mobile-First*: com o aumento expressivo do uso de dispositivos móveis, adotou-se uma abordagem que prioriza o desenvolvimento inicialmente para esse tipo de dispositivo, garantindo responsividade e melhor usabilidade em telas menores.

- Desenvolvimento *Cross-Platform*: visando ampliar o alcance das aplicações, surgiram soluções que permitem o desenvolvimento de uma única base de código capaz de ser executada em diferentes plataformas e sistemas operacionais, otimizando tempo e recursos de desenvolvimento.

- Integração com *APIs* e serviços externos: o consumo de *APIs* e a integração com sistemas de terceiros tornaram-se práticas essenciais para o fornecimento de conteúdo dinâmico, funcionalidades personalizadas e interoperabilidade entre sistemas diversos na *web*.

2.1.3 Exemplos

A aplicação dos conceitos de desenvolvimento *Front-End* pode ser observada em diversas plataformas amplamente utilizadas, que ilustram de forma prática os avanços recentes da área.

- **X (antigo Twitter)**: a plataforma é um exemplo de PWA, permitindo que o usuário instale uma versão *mobile* ou *desktop* diretamente a partir da interface *web*. O sistema destaca-se pelo alto grau de dinamismo e personalização de conteúdo com base nas preferências do usuário, além de apresentar uma interface altamente responsiva e interativa. (web.dev Editorial, 2019)

- **Nubank**: a fintech brasileira utiliza hoje o *toolkit* de desenvolvimento chamado *Flutter* para a criação do seu aplicativo que executa tanto em dispositivos Android quanto no iOS, dois sistemas operacionais totalmente distintos, porém que podemos utilizar o mesmo código-fonte para renderizar na tela os componentes necessários para visualização do mesmo (Nubank Editorial, 2021).

2.2 Arquitetura para Front-End

No desenvolvimento de software, a adoção de arquiteturas bem definidas é essencial para garantir que o sistema seja robusto, modular, testável e escalável. Essas arquiteturas estabelecem padrões organizacionais que orientam a estruturação dos arquivos e do código-fonte, promovendo consistência e facilitando a manutenção. No contexto do *Front-End*, tais práticas são igualmente relevantes, assegurando qualidade e previsibilidade no desenvolvimento da interface do usuário.

2.2.1 Histórico

Com o crescimento acelerado dos sistemas e o aumento da sua complexidade, tornou-se necessário adotar modelos arquiteturais mais eficientes, capazes de lidar com a evolução contínua dos projetos. Inicialmente, era comum o uso de sistemas monolíticos, nos quais todo o código — incluindo tanto o *Front-End* quanto o *Back-End* — era agrupado em um único repositório. Essa abordagem, embora funcional em projetos menores, demonstrou limitações à medida que os sistemas cresciam, dificultando a manutenção, a escalabilidade e a identificação de falhas. Como resposta a esses desafios, surgiram arquiteturas que promovem a modularização e o desacoplamento entre os componentes, permitindo que cada parte do sistema seja desenvolvida e gerida de forma independente.

2.2.2 Avanços recentes da área

Diversas evoluções têm sido observadas na arquitetura de sistemas *Front-End*, com destaque para duas abordagens fundamentais:

- **Componentização**: Trata-se de uma prática amplamente adotada por *frameworks JavaScript*, como os mencionados anteriormente, que consiste na construção de componentes atômicos, modulares e reutilizáveis. Esses componentes podem ser utilizados em diferentes partes do sistema ou até mesmo em outros projetos, por meio de pacotes e bibliotecas externas. Essa abordagem contribui significativamente para a manutenção, escalabilidade e padronização das interfaces.
- ***Microfrontends***: Essa arquitetura permite que equipes distintas desenvolvam, testem e implantem partes específicas da interface de forma independente. Cada equipe pode ser responsável por um componente ou módulo isolado, que, posteriormente, será integrado ao sistema como um todo. Essa abordagem é especialmente útil em grandes aplicações, onde o desacoplamento e a autonomia de desenvolvimento são fatores essenciais para a produtividade e organização do projeto.

2.2.3 Exemplos

Entre as arquiteturas modernas aplicadas ao *Front-End*, destaca-se o *Clean Architecture*, proposta por Robert C. Martin (Uncle Bob). Essa abordagem organiza o sistema em camadas concêntricas com regras explícitas de dependência, onde as camadas internas não possuem conhecimento das externas (MARTIN, 2012). Fundamentada nos princípios SOLID, o *Clean Architecture* prioriza a separação entre lógica de negócio e detalhes de implementação, promove a testabilidade por meio de interfaces bem definidas e assegura independência de frameworks, interface de usuário e mecanismos de persistência. No desenvolvimento de interfaces, essa arquitetura se traduz na construção de componentes autocontidos, com responsabilidades claramente delimitadas, distribuídos em camadas como: apresentação (componentes da interface), aplicação (casos de uso) e domínio (regras de negócio).

2.3 Migração de Sistemas Legados

A migração de sistemas legados consiste na adoção de técnicas e processos voltados à transição de uma aplicação existente para um novo ambiente tecnológico — seja esse relacionado à linguagem de programação, infraestrutura, banco de dados ou arquitetura de software. O principal objetivo desse processo é preservar as funcionalidades essenciais do sistema original, ao mesmo tempo em que se promove sua modernização. Para que essa transformação ocorra de forma segura e eficaz, é necessário, inicialmente, realizar um estudo completo do estado atual do sistema, avaliando seus requisitos funcionais e não funcionais. Com base nessa análise, define-se uma estratégia de migração que oriente as etapas de reestruturação, adaptação e implementação da nova solução.

2.3.1 Análise de projetos

2.3.1.1 Principais Definições

A análise de projetos compreende o estudo aprofundado de um sistema, considerando seus requisitos funcionais e não funcionais, bem como suas limitações, restrições e objetivos. Trata-se de uma etapa essencial no processo de migração de sistemas, pois permite compreender de forma estruturada o cenário atual da aplicação, identificar gargalos e definir os pontos que necessitam de melhorias. Essa etapa fornece subsídios para a tomada de decisão quanto às abordagens técnicas e estratégias mais adequadas para a reestruturação do software.

2.3.1.2 Histórico

Nos primeiros anos do desenvolvimento de sistemas, a elaboração de projetos era realizada de forma mais informal, sem o devido rigor metodológico. Muitas etapas fundamentais, como a análise de requisitos e definição clara de escopo, eram frequentemente negligenciadas, o que resultava em falhas no produto final, atrasos e dificuldades de manutenção. Com o avanço da engenharia de software, passou-se a adotar práticas mais sistematizadas, como o uso de documentação técnica, modelagem de requisitos e validação contínua. Esses avanços tornaram a análise de projetos uma etapa consolidada e indispensável para o sucesso de iniciativas de migração e modernização de sistemas.

2.3.1.3 Avanços recentes da área

Atualmente, observa-se a ampla adoção de metodologias ágeis no processo de desenvolvimento e manutenção de software. Essas metodologias, como Scrum e Kanban, promovem a colaboração entre os membros da equipe, a iteração contínua e a entrega incremental de funcionalidades. Sua aplicação contribui significativamente para a organização e execução de atividades de análise de projetos, desde a concepção até a manutenção do sistema, proporcionando maior flexibilidade e adaptabilidade às mudanças de requisitos ao longo do tempo.

2.3.1.4 Exemplos

- **Análise de documentação**: permite a identificação de pontos críticos e estruturas fundamentais no código-fonte do sistema, possibilitando a compreensão detalhada de seu funcionamento em nível técnico e lógico.
- **Análise de protótipos**: por meio da avaliação dos protótipos desenvolvidos pela equipe de UX/UI, torna-se possível compreender funcionalidades e interações esperadas do sistema, além de auxiliar na padronização de elementos visuais, como cores, espaçamentos e tamanhos de fonte, por meio do uso de *design tokens*.

2.3.2 Estratégias para migração de sistemas

2.3.2.1 Principais Definições

A migração de sistemas tem como principal objetivo adaptar aplicações legadas a novas plataformas tecnológicas sem comprometer as funcionalidades já existentes. Esse processo, geralmente, é dividido em três fases principais: a engenharia reversa (responsável pela compreensão do sistema legado), as alterações estruturais (como refatoração e reorganização do código e dos dados) e a engenharia direta (reconstrução do sistema com base nos novos requisitos e padrões) (LIM; TAN, 2025).

2.3.2.2 Histórico

Historicamente, os processos de migração eram conduzidos de forma isolada, com forte dependência do conhecimento tácito dos profissionais envolvidos. A ausência de documentação adequada nos sistemas legados dificultava o entendimento de sua estrutura interna, tanto no que se refere ao código-fonte quanto à sua arquitetura. Com o amadurecimento da área, surgiram abordagens mais estruturadas que sistematizam o processo, como os modelos em ferradura, híbrido e SCORE/RM. Esses modelos formalizam as etapas da migração, integrando atividades de análise, reestruturação e testes, o que permite conduzir migrações com maior segurança, rastreabilidade e consistência, minimizando o risco de perdas funcionais ou falhas durante a transição (LIM; TAN, 2025).

2.3.2.3 Avanços recentes da área

- **Automação de Migração**: o uso de ferramentas especializadas e *scripts* automatizados tem ganhado destaque, reduzindo significativamente o trabalho manual envolvido no processo de migração. Essa automação contribui também para a minimização de erros humanos, aumentando a confiabilidade e a eficiência das transições.
- **Uso de Contêineres e Orquestração**: tecnologias como Docker e Kubernetes têm sido amplamente adotadas em migrações arquiteturais, facilitando a decomposição de sistemas monolíticos em microserviços. Além disso, essas ferramentas otimizam o processo de implantação contínua em ambientes baseados em nuvem, promovendo maior escalabilidade e controle operacional.
- **Abordagens Incrementais e Iterativas**: em substituição ao modelo tradicional de substituição total do sistema (conhecido como *Big Bang*), diversas organizações têm optado por estratégias incrementais ou modulares. Esse tipo de abordagem permite mitigar riscos, possibilitando a manutenção parcial da operação do sistema durante o processo de migração (LIM; TAN, 2025).

2.3.2.4 Exemplos

- **Migração de Banco de Dados**: em busca de maior flexibilidade e escalabilidade, muitos sistemas que originalmente utilizavam bancos de dados relacionais têm sido migrados para modelos NoSQL. Essa mudança é especialmente vantajosa em arquiteturas distribuídas e ambientes em nuvem, onde há demanda por armazenar dados semiestruturados de maneira eficiente (LIM; TAN, 2025).
- **Migração para a Nuvem**: diversas empresas que ainda operam com servidores locais (*on-premises*) têm adotado estratégias de migração para a nuvem. Esse processo envolve a transferência de dados, aplicações e demais componentes, com o

objetivo de explorar os benefícios de escalabilidade, elasticidade e redução de custos proporcionados por ambientes em nuvem.

2.4 Experiência do Usuário (UX)

2.4.1 Principais Definições

A Experiência do Usuário (UX) refere-se à qualidade da interação entre o usuário e um sistema, abrangendo aspectos como usabilidade, acessibilidade, eficiência, desempenho e satisfação geral durante a utilização da interface. Essa experiência é influenciada não apenas pelo aspecto visual, mas por toda a jornada de interação com o produto ou serviço, sendo determinante para o engajamento e a aceitação da solução digital.

No contexto educacional, (PICOLO, 2021) destaca que a aplicação de princípios de UX em sistemas acadêmicos resulta em melhorias significativas na usabilidade e acessibilidade, contribuindo diretamente para uma interação mais intuitiva e eficaz entre estudantes e plataformas digitais.

2.4.2 Histórico

As origens da Experiência do Usuário remontam a áreas como psicologia, *design* e ergonomia, nas quais se busca compreender como os indivíduos percebem, interagem e reagem aos estímulos do ambiente — no caso, a interface de um sistema. Inicialmente, os estudos focavam predominantemente na aparência visual (*design* gráfico). No entanto, com o avanço das tecnologias e a crescente exigência por soluções mais centradas nas necessidades humanas, o conceito expandiu-se para além do visual, abrangendo toda a jornada do usuário. Com isso, surgiu a integração entre UX e UI, consolidando-se como uma abordagem conjunta e estratégica no desenvolvimento de sistemas interativos.

2.4.3 Avanços recentes da área

O Design Centrado no Usuário (DCU) tem ganhado destaque como uma abordagem fundamental no desenvolvimento de sistemas, priorizando a experiência do usuário em todas as etapas do processo. Essa filosofia reformula a forma como são concebidos os protótipos e os elementos de *design*, buscando alinhar as soluções criadas às reais necessidades, expectativas e comportamentos dos usuários finais.

2.4.4 Exemplos

Um exemplo notável da aplicação bem-sucedida dos princípios de experiência do usuário é a empresa Apple. A companhia investe fortemente na criação de produtos com

alto nível de coesão visual e funcional, garantindo uma interface intuitiva e uma navegação fluida. Tal compromisso com a usabilidade e o refinamento estético consolidou a Apple como uma das principais referências na área.

2.5 Segurança em Sistemas de Informação

2.5.1 Principais Definições

A segurança em sistemas de informação é um componente essencial no desenvolvimento de soluções tecnológicas, pois visa proteger os dados e garantir o funcionamento confiável dos sistemas. No contexto deste trabalho, a segurança é indispensável para assegurar a integridade e a confiabilidade das informações manipuladas. Ela abrange métodos, tecnologias e políticas voltadas à prevenção de ataques e à proteção da confidencialidade, integridade e disponibilidade dos dados — princípios fundamentais conhecidos como a tríade *Confidentiality, Integrity, Availability (CIA)*. A proteção pode ser implementada em diferentes níveis, incluindo aspectos físicos, lógicos e organizacionais, promovendo uma abordagem sistêmica e abrangente da segurança da informação.

2.5.2 Histórico

Historicamente, as preocupações com a segurança da informação surgiram em contextos militares e governamentais, onde o sigilo e a proteção de dados estratégicos eram prioritários. Nesse período, foram desenvolvidas técnicas e ferramentas voltadas à prevenção de acessos não autorizados e à manutenção da confidencialidade. Com o avanço da tecnologia e a crescente digitalização de dados, as ameaças se diversificaram, tornando necessária a adoção de mecanismos mais sofisticados e distribuídos de segurança. Atualmente, sistemas modernos empregam práticas avançadas, como autenticação multifator, criptografia, auditoria contínua e monitoramento em tempo real, de modo a mitigar vulnerabilidades e oferecer maior resistência a ataques cibernéticos.

2.5.3 Avanços recentes da área

- Inteligência Artificial (IA) e *Machine Learning (ML)*: O uso de técnicas de IA e aprendizado de máquina tem se destacado como um avanço significativo na área de segurança da informação. Tais tecnologias são aplicadas na detecção de anomalias, comportamentos suspeitos e na automatização de processos, proporcionando uma camada adicional de proteção que interpreta padrões e ameaças de maneira adaptativa e contextual, simulando características cognitivas humanas.
- *Principle of Least Privilege (POLP)*: O princípio do privilégio mínimo define que usuários, programas ou processos devem operar com o menor nível de acesso neces-

sário para executar suas funções. Essa abordagem limita o escopo de ações disponíveis a cada entidade, reduzindo a superfície de ataque em caso de comprometimento. Além disso, contribui para o confinamento de danos potenciais, impedindo que ameaças se espalhem por todo o sistema (LORD, 2023).

2.5.4 Exemplos

- Tokens de autenticação: Mecanismos utilizados para validar de forma segura as identidades dos usuários, possibilitando que apenas clientes autenticados realizem requisições ao servidor (GeeksforGeeks, 2023).
- Firewalls e antivírus: Soluções que atuam na proteção de redes e dispositivos, impedindo acessos não autorizados e bloqueando softwares maliciosos por meio da análise e controle do tráfego de dados.
- Criptografia: Técnica de segurança que transforma dados sensíveis em formatos codificados, assegurando sua confidencialidade e integridade tanto durante a transmissão quanto no armazenamento.
- *Multi-Factor Authentication (MFA)*: Estratégia de verificação que exige dois ou mais métodos de autenticação distintos, como senha e biometria, aumentando a robustez no controle de acesso a sistemas.

2.6 Versionamento de Código

2.6.1 Principais Definições

O versionamento de código é uma prática fundamental na engenharia de software, utilizada para controlar e gerenciar diferentes versões de arquivos, principalmente códigos-fonte. Por meio de repositórios — locais físicos ou virtuais — é possível armazenar, recuperar e comparar modificações, além de facilitar o trabalho colaborativo entre desenvolvedores, promovendo segurança, rastreabilidade e organização no desenvolvimento de sistemas.

Segundo (ZOLKIFLI; NGAH; DERAMAN, 2018), os sistemas de controle de versão (VCS, do inglês *Version Control Systems*) são indispensáveis para equipes modernas, pois mantêm um histórico completo das alterações realizadas, reduzem o risco de perda de dados e simplificam o desenvolvimento em ambientes colaborativos. Entre as soluções mais utilizadas, destaca-se o *Git*, sistema distribuído que proporciona maior autonomia e robustez ao processo de versionamento.

2.6.2 Histórico

A necessidade de versionamento de arquivos surgiu com o crescimento dos projetos de software em equipe, que exigiam um controle rigoroso das modificações realizadas por múltiplos colaboradores. Inicialmente, os processos eram realizados manualmente, com armazenamento local de cópias datadas dos arquivos. Posteriormente, surgiram sistemas de controle de versão centralizados, como o *CVS* e o *Subversion (SVN)*, que permitiram um gerenciamento mais eficiente, porém ainda com dependência de um servidor central. A evolução para sistemas distribuídos, como o *Git*, criado por Linus Torvalds em 2005, representou um marco significativo, ao possibilitar que cada colaborador possuísse uma cópia completa do histórico do projeto, promovendo maior autonomia, segurança e flexibilidade.

2.6.3 Avanços Recentes na Área

O desenvolvimento do *Git* consolidou práticas modernas de versionamento, como o trabalho em ramos (*branches*), fusões (*merges*) e revisões de código (*code reviews*). Avanços recentes incluem a integração de sistemas de versionamento a plataformas de hospedagem como *GitHub*, *GitLab* e *Bitbucket*, que agregam funcionalidades de integração contínua (CI), entrega contínua (CD), automações de testes e ferramentas de gerenciamento de projetos. Recursos como *GitHub Actions* e *GitLab CI/CD* exemplificam como o versionamento tornou-se parte integrante de processos de desenvolvimento ágil e DevOps, aumentando a qualidade e a eficiência das entregas de software.

2.6.4 Exemplos

Embora associado majoritariamente ao desenvolvimento de software, o conceito de versionamento é aplicável em diversos contextos. Em ambientes como o *Google Drive* e a suíte *Microsoft Office*, por exemplo, é possível manter diferentes versões de documentos por meio de salvamentos sucessivos. Contudo, no desenvolvimento de software, o uso de sistemas especializados é essencial. O *GitHub* se destaca como a plataforma pública mais popular para hospedagem e colaboração em projetos de código aberto. O *GitLab*, por sua vez, foi utilizado no presente projeto para gerenciar o repositório do *Front-End* modernizado da plataforma *Classroom eXperience*, possibilitando controle rigoroso das alterações, organização do fluxo de trabalho e integração com *pipelines* de automação de testes e *builds*.

2.7 Documentação e Transferência de Conhecimento

2.7.1 Principais Definições

A documentação de um sistema é um elemento essencial para a organização e manutenção de sua estrutura. Ela garante que as informações relacionadas ao projeto — como suas características, processos, fórmulas e demais aspectos relevantes — estejam devidamente registradas de forma escrita, digital, visual ou em código. Além disso, a transferência de conhecimento deve ocorrer de maneira clara e transparente, permitindo que colaboradores atuais ou futuros compreendam e deem continuidade ao desenvolvimento do sistema de forma eficaz e estruturada.

2.7.2 Histórico

A prática de documentar projetos sempre teve papel fundamental em diversas áreas do conhecimento, e na computação não é diferente. Desde os primeiros sistemas computacionais, a necessidade de registrar informações sobre arquitetura, funcionamento e manutenção foi reconhecida como fator crucial para a continuidade e evolução dos projetos. Com os avanços tecnológicos, surgiram ferramentas especializadas que tornaram esse processo mais ágil e acessível, possibilitando a centralização e digitalização dos registros. Atualmente, a documentação está amplamente disponível sob demanda, facilitando a colaboração entre equipes e mitigando os desafios tradicionalmente enfrentados na preservação e disseminação do conhecimento técnico.

2.7.3 Avanços recentes da área

- Ferramentas de Documentação Colaborativa: As plataformas colaborativas possibilitam que múltiplos usuários editem simultaneamente, adicionem comentários e acompanhem modificações em tempo real. Essa abordagem promove um ambiente de trabalho mais integrado, garantindo alinhamento entre os membros da equipe quanto aos objetivos do projeto e às especificações técnicas (INBEC, 2024).

- Automação na Documentação de Sistemas: A automação aplicada à documentação de software tem se mostrado uma solução eficiente para reduzir o esforço manual e garantir a atualização contínua dos registros técnicos. Ferramentas especializadas são capazes de gerar automaticamente documentação a partir do código-fonte, extraindo descrições de funções, classes e interfaces de programação (APIs). Algumas soluções, inclusive, utilizam inteligência artificial para organizar essas informações e propor melhorias, seguindo padrões estabelecidos de padronização e clareza.

2.7.4 Exemplos

- **ClickUp**: Plataforma de gerenciamento de projetos que oferece funcionalidades voltadas à documentação colaborativa, como edição em tempo real, integração com tarefas e armazenamento centralizado de informações, facilitando a organização dos registros técnicos ao longo do ciclo de vida do software.
- **Swagger**: Conjunto de ferramentas de código aberto voltado ao *design*, construção e documentação de APIs REST, permitindo a criação automatizada e padronizada de contratos de API e sua respectiva documentação.

3 Trabalhos Relacionados

A modernização de sistemas *Front-End* tem-se consolidado como uma demanda recorrente diante da evolução acelerada das tecnologias *Web* e da crescente exigência por interfaces modernas, responsivas e integradas a serviços externos. Esse cenário reflete o aumento expressivo do uso de aplicações digitais e da demanda por soluções otimizadas para múltiplas plataformas, especialmente dispositivos móveis, que atualmente representam a maior parte do consumo de conteúdo na *web* (LUFT; ROSENFELDER; SYDOW, 2021). Além disso, observa-se no mercado brasileiro um crescimento significativo na procura por desenvolvedores especializados em tecnologias voltadas à *web* responsiva e *mobile*, reforçando a importância da modernização como resposta às exigências do setor (FAETERJ-RIO, 2023). Nesse contexto, o processo de modernização vai além da substituição tecnológica, exigindo reestruturações arquiteturais e práticas voltadas à escalabilidade, manutenibilidade e experiência do usuário, como proposto neste trabalho com a plataforma *Classroom eXperience*.

Raksi explora esse cenário por meio de um estudo de caso aplicado à atualização de um sistema corporativo do setor financeiro, enfatizando a importância de arquiteturas evolutivas, componentes desacoplados e migrações incrementais para garantir a continuidade dos serviços e a mitigação de riscos (RAKSI, 2017). A autora destaca que a modernização exige não apenas transformação tecnológica, mas também reestruturação da arquitetura das aplicações e alinhamento entre as áreas de negócio e a tecnologia.

No contexto acadêmico, Picolo apresenta a modernização do SIGA — Sistema de Gestão Acadêmica — abordando a refatoração de seu *Front-End* e propondo melhorias na usabilidade e acessibilidade da interface (PICOLO, 2021). Tal proposta possui similaridades diretas com o escopo do presente trabalho, reforçando a relevância da modernização tecnológica em ambientes institucionais de ensino.

De forma complementar, Leon e Horita (LEON; HORITA, 2020) propõem uma visão estruturada sobre a modernização de sistemas sob a ótica da transformação digital, enfatizando que a adoção de arquiteturas modernas, como microsserviços, é fundamental para garantir escalabilidade, flexibilidade e aderência às demandas organizacionais contemporâneas. Os autores destacam que a modernização arquitetural vai além de uma simples substituição tecnológica, configurando-se como um processo estratégico que requer alinhamento entre os objetivos de negócio e as capacidades técnicas. Essa abordagem fornece uma base teórica relevante para as decisões adotadas no redesenho da plataforma *Classroom eXperience*, especialmente ao considerar os desafios impostos pela arquitetura monolítica anteriormente utilizada.

Sousa apresenta o redesenho do EducaTea, voltado ao público infantil com Transtorno do Espectro Autista. Embora o foco seja a inclusão, o projeto aborda diretamente práticas de modernização da interface, utilizando *gamificação* e *design* centrado no usuário (SOUSA, 2024). O uso de diretrizes como o Guia de Acessibilidade de Interfaces para Autismo (GAIA) aproxima-se da abordagem adotada no presente projeto em termos de melhoria da experiência do usuário.

Clulow e Brace-Govan, por sua vez, investigam o impacto das tecnologias *Web* na educação, evidenciando que sistemas bem estruturados e baseados na experiência do usuário contribuem significativamente para o engajamento e a aprendizagem (CLULOW; BRACE-GOVAN, 2003). Seus achados apoiam a proposta do CX em reforçar a interação entre aluno e conteúdo através de uma interface moderna e adaptável.

Por fim, o artigo (SODRÉ et al., 2021) discute os avanços tecnológicos na educação e como estes impactam diretamente a estrutura e a eficácia dos ambientes virtuais de aprendizagem. A adoção de novas ferramentas digitais e plataformas de interação reforça a importância da atualização tecnológica nos sistemas educacionais, conectando-se diretamente com a proposta deste trabalho.

Dessa forma, os trabalhos aqui discutidos convergem ao apontar que a modernização tecnológica — seja em sistemas empresariais ou educacionais — demanda mais do que a substituição de tecnologias obsoletas. Exige-se uma visão integrada que considere a arquitetura, a experiência do usuário e a viabilidade de manutenção e evolução do projeto. O presente trabalho se insere nesse cenário ao propor uma evolução concreta da plataforma *Classroom eXperience*, fundamentada nas boas práticas destacadas pela literatura técnica e acadêmica.

4 Desenvolvimento

O desenvolvimento do sistema teve início com o planejamento das atividades a serem realizadas ao longo do projeto. Nesta etapa inicial, grande parte do esforço concentrou-se na identificação, análise e organização das informações presentes no sistema atual, no repositório de código e no protótipo desenvolvido no Figma da plataforma CX. Com essa base estabelecida, foi possível avançar para a definição dos requisitos funcionais e não funcionais, bem como para a modelagem dos dados. Por fim, definiram-se as tecnologias a serem utilizadas, dando início à etapa de implementação do sistema.

4.1 Planejamento

Na fase de planejamento, foi realizada a identificação, análise e organização das informações fundamentais para o desenvolvimento do projeto. Por se tratar de uma migração, foi possível extrair dados relevantes a partir do sistema já existente, tais como elementos da modelagem de dados, requisitos funcionais e propostas de interface do usuário, entre outros aspectos essenciais para embasar a reestruturação da solução.

Diante desse cenário, o planejamento foi estruturado em três frentes principais: **Análise do Estado Atual do Sistema**, **Requisitos do Sistema** e **Modelagem de Dados e Contratos de API**.

4.1.1 Análise do Estado Atual do Sistema

A análise do estado atual do sistema foi fundamental para proporcionar uma visão clara dos elementos que deveriam ser migrados e desenvolvidos. Para isso, foram examinados o site do CX — disponível em: http://cx.facom.ufu.br/ —, seu banco de dados e o respectivo código-fonte.

Com base na interface atual do site e no projeto elaborado no Figma, foi possível planejar o novo visual do sistema, incluindo os estilos e componentes a serem implementados. Paralelamente, essa análise também permitiu a extração de informações relevantes para a compreensão do funcionamento do sistema e de seus requisitos.

A análise do banco de dados e do código-fonte foi essencial para a consolidação dos requisitos do sistema e das entidades utilizadas no *Front-End*. Como o desenvolvimento foi realizado em conjunto com o projeto do *Back-End*, observou-se uma atenção maior à modelagem de dados nessa parte da aplicação. Dessa forma, o foco principal do presente trabalho concentrou-se na identificação e organização dos requisitos funcionais e não funcionais, bem como na definição dos contratos de comunicação com a API — isto

é, os modelos que definem a estrutura das mensagens trocadas por meio de requisições HTTP entre o *Front-End* e o *Back-End*. Em seguida, foram implementadas as entidades específicas voltadas ao uso interno da aplicação.

4.1.2 Requisitos do Sistema

Os requisitos do sistema representam as especificações das funcionalidades e das restrições que o software deve cumprir para atender às necessidades e objetivos definidos para o projeto. Esses requisitos são classificados em duas categorias principais: Requisitos Funcionais (RF) e Requisitos Não-Funcionais (RNF).

Neste trabalho, os requisitos foram definidos com base em uma análise detalhada das funcionalidades existentes no sistema original, bem como com o apoio do professor orientador, que auxiliou na identificação daqueles considerados mais relevantes para a migração e modernização da aplicação.

4.1.2.1 Requisitos Funcionais

Os requisitos funcionais estão diretamente relacionados às regras de negócio do sistema, representando as funcionalidades que têm como objetivo atender às necessidades dos usuários e solucionar os problemas propostos. Com base na análise do sistema atual e nos objetivos do projeto, foram definidos os seguintes requisitos funcionais:

- RF01: Cadastro de usuário (aluno) — o usuário deve poder se registrar no sistema, fornecendo suas informações para acesso como aluno;
- RF02: *Login* — o usuário deve conseguir autenticar sua sessão utilizando e-mail e senha;
- RF03: Redefinição de senha — o usuário deve ser capaz de redefinir sua senha por meio do e-mail previamente registrado no sistema;
- RF04: *Logout* — o usuário deve ter a opção de encerrar sua sessão atual, retornando para a tela de *login*;
- RF05: Alteração de idioma — o usuário deve poder alterar o idioma da interface a qualquer momento;
- RF06: Carregamento de dados do usuário — ao iniciar a sessão, o sistema deve carregar as informações básicas do usuário para fins de personalização e funcionamento adequado;
- RF07: Carregamento de notificação do ILS diário — o sistema deve notificar o usuário diariamente com a solicitação de preenchimento do ILS;

- RF08: Registro do ILS diário — o usuário deve poder responder às perguntas do ILS diário diretamente pela notificação;
- RF09: Notificações de comentários recentes — o sistema deve exibir notificações relacionadas a comentários recentes feitos em conteúdos relevantes;
- RF10: Notificações de novos *quizzes* — o sistema deve informar o usuário sobre novos *quizzes* disponibilizados;
- RF11: Listagem de disciplinas — o usuário deve conseguir visualizar uma lista de disciplinas categorizadas como Matriculadas, Meu Curso e Públicas;
- RF12: Matrícula em disciplina — o usuário deve ser capaz de se matricular em uma disciplina disponível;
- RF13: Acesso a detalhes da disciplina — o usuário deve conseguir visualizar as informações detalhadas de uma disciplina ao visitar sua página;
- RF14: Acesso às aulas da disciplina — o usuário deve poder acessar as aulas associadas à disciplina, independentemente do formato (vídeo, slides ou completo);
- RF15: Edição de dados pessoais — o usuário deve poder atualizar seus dados cadastrais diretamente no sistema;
- RF16: Registro de contexto de acesso — o usuário deve poder registrar novos contextos de acesso conforme sua necessidade;
- RF17: Visualização de contexto de acesso — o usuário deve conseguir consultar os contextos de acesso previamente registrados para fins de personalização;
- RF18: Registro de restrições — o sistema deve permitir que o usuário defina restrições de acesso a determinados recursos;
- RF19: Listagem de restrições — o usuário deve poder visualizar todas as restrições que definiu anteriormente;
- RF20: Exclusão de restrições — o sistema deve permitir ao usuário remover restrições de acesso previamente criadas;
- RF21: Registro de preferências — o usuário deve ser capaz de registrar suas preferências individuais de uso do sistema;
- RF22: Visualização do questionário ILS completo — o sistema deve permitir ao usuário acessar todas as perguntas do questionário ILS;
- RF23: Resposta ao questionário ILS completo — o usuário deve conseguir responder integralmente às perguntas do ILS;

- RF24: Listagem de *badges* — o sistema deve permitir ao usuário visualizar todas as conquistas (*badges*) obtidas até o momento.

4.1.2.2 Requisitos Não-Funcionais

Os requisitos não funcionais descrevem as características de qualidade do sistema, ou seja, como ele deve se comportar e operar, independentemente das funcionalidades específicas. Abaixo, são apresentados os requisitos definidos para este projeto:

- RNF01: Conectividade à internet — o sistema deve disponibilizar todas as suas funcionalidades quando conectado à rede;
- RNF02: Portabilidade — o sistema deve estar acessível em diferentes navegadores, dispositivos e sistemas operacionais, garantindo ampla compatibilidade;
- RNF03: Interface intuitiva, responsiva e acessível — o sistema deve possuir uma interface clara e de fácil utilização, responsiva a diferentes tamanhos de tela, e com conteúdo acessível a todos os usuários, inclusive aqueles com limitações ou deficiências;
- RNF04: Segurança e uso de sessões de usuário — o sistema deve garantir a proteção dos dados dos usuários, autenticando e mantendo sessões seguras durante o uso;
- RNF05: Disponibilidade e integridade dos dados — os dados armazenados devem estar sempre disponíveis para os usuários e devem ser protegidos contra modificações não autorizadas;
- RNF06: Modularização e documentação do código — o código-fonte deve ser estruturado de forma modular, visando facilitar a organização, manutenção e testes. A documentação associada deve seguir boas práticas e normas técnicas;
- RNF07: Internacionalização — o sistema deve oferecer suporte a múltiplos idiomas, permitindo a adaptação de sua interface para diferentes localidades.

4.1.3 Modelagem de dados e contratos da API

Conforme mencionado anteriormente, a modelagem de dados principal foi conduzida pela equipe responsável pelo *Back-End*. Sendo assim, este projeto ficou encarregado do planejamento e da organização das entidades internas do sistema, adequando-as aos dados disponibilizados pela API. Tais entidades correspondem a objetos e classes utilizados no *Front-End*, cuja adoção segue o paradigma da programação orientada a objetos, promovendo maior modularização, legibilidade e organização do código.

Os contratos de API, por sua vez, consistem em modelos que padronizam o envio e o recebimento de dados entre cliente e servidor. Eles definem a estrutura das requisições e respostas, especificando parâmetros, corpo da requisição, formato da resposta e outros comportamentos esperados. Essa padronização é essencial para garantir a comunicação correta e eficiente entre as diferentes camadas do sistema.

A seguir, são apresentados dois exemplos de contratos criados durante a fase de planejamento do projeto:

4.1.3.1 Exemplo de Contrato 1 - *Login* de Usuário

```
Endpoint: POST "/api/auth/login"

Request Body:
    {
        "email": "email@email.com",
        "password": "password123"
    }

Request Response:
    {
        "access_token": "367412897630912",
        "expire_date": "2025-01-14T19:46:40.291Z"
    }
```

4.1.3.2 Exemplo de Contrato 2 - Buscar Disciplinas por Curso

```
Endpoint: GET "/api/subjects"

Request Query Parameters:
    {
        "course_id": 1
    }

Request Response:
    [
        {
            "id": 4,
            "title": "Desenvolvimento Web",
            "teacher": {
                "id": 4,
                "name": "Prof. D"
```

```
            }
        }
    ]
```

Além dos contratos de API, o sistema também foi modelado com base em uma estrutura de camadas composta por *Models*, *Entities* e *Mappers*. Essa modelagem foi fundamental para garantir a organização e a consistência dos dados entre as camadas da aplicação, sendo detalhada posteriormente na seção de implementação do sistema.

4.2 Tecnologias Utilizadas

Dado que este trabalho tem como objetivo a modernização do sistema CX, foram selecionadas tecnologias distintas daquelas anteriormente empregadas no projeto original, com a intenção de manter a funcionalidade existente e, ao mesmo tempo, incorporar novas características e melhorias.

Como o foco principal está voltado para a modernização da camada de *Front-End*, a maior parte das tecnologias descritas nesta seção está relacionada à construção da interface do usuário, à sua estilização e ao funcionamento interno da aplicação na camada de apresentação.

4.2.1 Angular

O Angular é um *framework Web* de código aberto desenvolvido pelo Google, amplamente utilizado para a construção de aplicações *web* interativas e com conteúdo dinâmico. Ele é baseado em TypeScript (TS), um superconjunto do JS, cuja definição será abordada na subseção seguinte.

A principal abordagem do Angular consiste na componentização da interface, incentivando o desenvolvedor a organizar a aplicação em componentes menores e reutilizáveis, que podem ser combinados para formar a interface final. Cada componente é responsável por sua própria estrutura e funcionamento, sendo composto, geralmente, por três arquivos principais: um arquivo com a estrutura em HTML, outro com a estilização em CSS e um terceiro com as lógicas de comportamento em TS. Ressalta-se que, para estilização, é possível utilizar não apenas CSS, mas também outras folhas de estilo, como Sassy Cascading Style Sheets (SCSS) ou Syntactically Awesome Style Sheets (SASS). Para mais informações, considere acessar sua página oficial.

Adicionalmente, o Angular dispõe de uma poderosa ferramenta de linha de comando (CLI), que auxilia o desenvolvedor na criação de novos componentes, serviços e outros elementos, promovendo a padronização do código e a automação de processos. Diversos outros recursos podem ser consultados diretamente em sua documentação.

No desenvolvimento deste projeto, foi utilizada a versão **18.2.0** do Angular, que traz melhorias importantes em desempenho, modularização e integração com bibliotecas modernas.

4.2.2 TypeScript

O TypeScript é um superconjunto do JavaScript, ou seja, trata-se de uma linguagem que acrescenta novos recursos e ferramentas ao JavaScript, tornando o desenvolvimento mais eficiente, robusto, menos suscetível a erros e com maior facilidade de manutenção. Mesmo com essas adições, mantém total compatibilidade com o JavaScript tradicional (também chamado de *Vanilla* JavaScript, que se refere ao código escrito puramente em JavaScript, sem bibliotecas ou *frameworks* adicionais).

Entre as principais funcionalidades oferecidas pelo TypeScript, destacam-se a tipagem estática e o suporte à programação orientada a objetos, recursos que não estão disponíveis nativamente no JavaScript. Tais características proporcionam maior controle sobre o código e facilitam a detecção de falhas ainda na fase de desenvolvimento.

O código escrito em TypeScript pode ser executado tanto no lado do cliente quanto no lado do servidor. No entanto, antes de sua execução, ele passa por um processo denominado *transpilação*, no qual os arquivos com extensão `".ts"` são convertidos em arquivos JavaScript (`.js`). Esse processo garante a compatibilidade com ambientes e bibliotecas que utilizam exclusivamente JavaScript. Dessa forma, o TypeScript é empregado exclusivamente na etapa de desenvolvimento, enquanto a execução ocorre em JavaScript.

No escopo deste projeto, utilizou-se a versão **5.5.2** do TypeScript. A documentação completa encontra-se disponível em www.typescriptlang.org.

4.2.3 Angular Material

O Angular Material é uma biblioteca de componentes de código aberto que implementa, para aplicações desenvolvidas com Angular, os princípios do sistema de *design* criado pelo Google, o Material Design. Essa biblioteca oferece uma ampla variedade de componentes de interface prontos para uso, como botões, menus, formulários, cartões e tabelas, permitindo sua integração direta ao código de forma padronizada e responsiva.

Entre suas principais características destacam-se a personalização visual, a padronização de estilos, a agilidade no desenvolvimento de interfaces, a melhoria na qualidade do código e a versatilidade na construção de aplicações com aparência profissional e consistente. Ao seguir diretrizes de *design* bem estabelecidas, o Angular Material contribui significativamente para a uniformidade da experiência do usuário em diferentes partes do sistema.

Neste trabalho, foi adotada a versão baseada no **Material Design 3**, cujas diretrizes completas podem ser consultadas em sua documentação oficial.

4.2.4 Interface de Programação de Aplicação (API)

As Interfaces de Programação de Aplicações, ou APIs (*Application Programming Interfaces*), consistem em mecanismos que definem um conjunto de protocolos, normas e ferramentas utilizadas para permitir a comunicação entre dois sistemas de forma padronizada. Por meio das definições de uma API, é possível consumir seus serviços sem que seja necessário conhecer sua implementação interna, bastando entender suas entradas e saídas esperadas — como exemplificado anteriormente nos contratos de API.

Essas interfaces contribuem para a simplificação do desenvolvimento, favorecendo a reutilização de funcionalidades e acelerando o processo de criação e manutenção de serviços. Seu valor é especialmente relevante no contexto atual, em que empresas exigem agilidade nas entregas, atualizações contínuas e escalabilidade de suas soluções.

As APIs estão presentes em praticamente todas as áreas da tecnologia, servindo como base para aplicações de pequeno a grande porte. Elas podem ser públicas ou privadas, utilizadas para leitura, escrita ou modificação de dados, entre outros propósitos. Um exemplo representativo é a API do Google Maps, que pode ser integrada a diferentes sistemas, tanto visualmente — como no próprio site <www.google.com/maps> — quanto por meio de requisições diretas ao seu *endpoint*, permitindo que desenvolvedores consumam seus dados e funcionalidades conforme as políticas de uso estabelecidas.

4.3 Implementação do Sistema

4.3.1 Arquitetura Limpa

Antes do início do desenvolvimento do sistema, foi necessário planejar como seria realizada a organização de seus componentes, a interação entre os diferentes módulos e a estruturação geral do repositório. Para isso, foram aplicadas técnicas de Arquitetura de Software, sendo adotada a abordagem conhecida como *Clean Architecture* (Arquitetura Limpa), proposta por Robert C. Martin em sua obra *Arquitetura Limpa: o Guia do Artesão para Estrutura e Design de Software* (Robert C. Martin, 2021).

A Arquitetura Limpa se baseia em princípios fundamentais que visam a construção de um código robusto, testável e desacoplado de tecnologias externas. Entre os princípios adotados, destacam-se: a separação de responsabilidades, a regra da dependência, a testabilidade e a independência de plataforma. Esse modelo arquitetural organiza o sistema em camadas concêntricas, conhecidas como módulos, nas quais cada camada possui uma responsabilidade bem definida e interage com as demais de forma controlada.

No contexto deste projeto, a estrutura foi dividida de forma que os módulos principais fossem organizados dentro do diretório *source*, mostrado pela Figura 1, onde se concentram os arquivos relacionados ao desenvolvimento da aplicação. Essa estrutura favorece a manutenibilidade e a escalabilidade do sistema, promovendo a reutilização de componentes e facilitando a identificação de responsabilidades em cada parte do código.

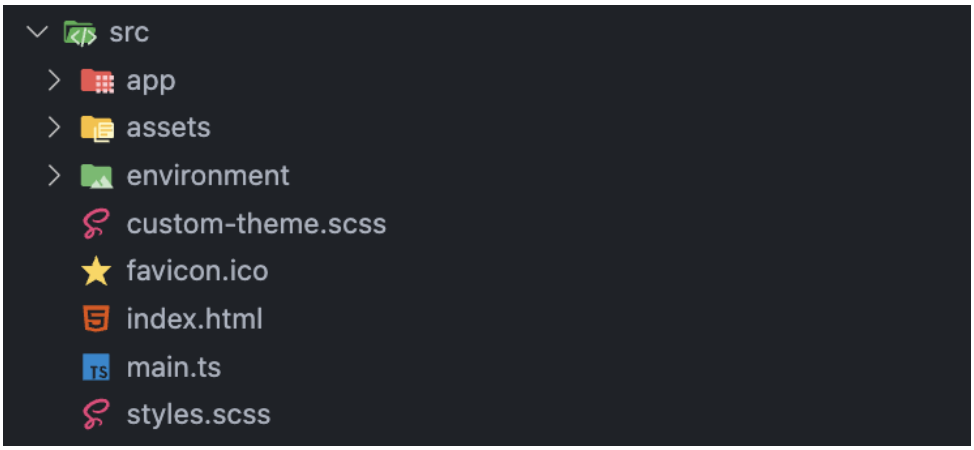


Figura 1 – Diretório `src`

A estrutura inicial do projeto contempla três diretórios principais, responsáveis pela organização do código-fonte e demais recursos da aplicação:

1. `app`: contém o código-fonte da aplicação;
2. `assets`: armazena arquivos estáticos, como imagens, ícones, logotipos e *gifs*;
3. `environments`: responsável pelos arquivos de configuração de ambiente (por exemplo, produção e desenvolvimento).

Os diretórios `assets` e `environments` apresentam funções autoexplicativas e não exigem maiores aprofundamentos. Desta forma, o foco da análise será direcionado ao diretório `app`, mostrado pela Figura 2, onde se encontra o núcleo da aplicação e onde a arquitetura adotada está devidamente estruturada.

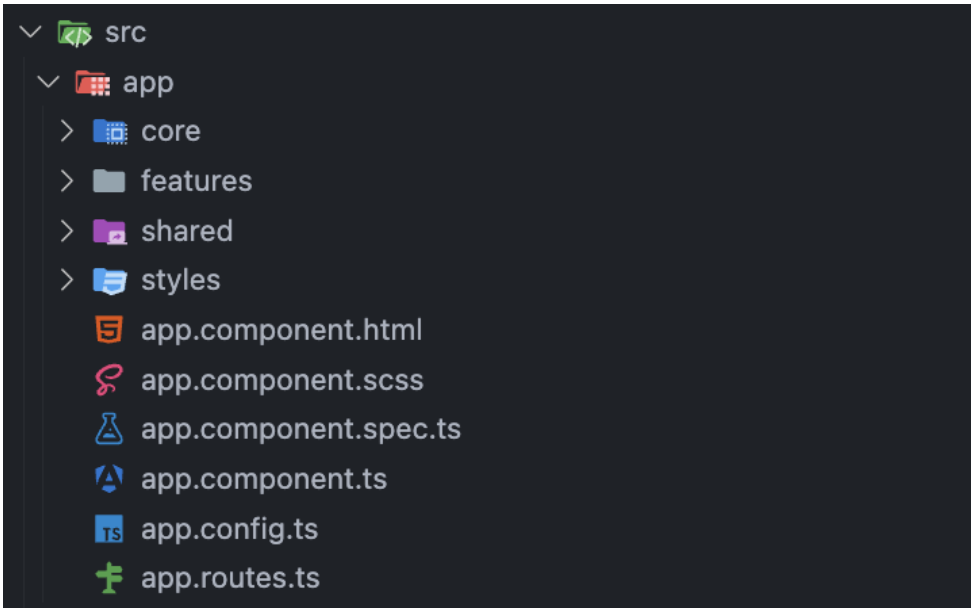


Figura 2 – Diretório `app`

Podemos verificar que existem quatro diretórios e os arquivos base da aplicação. Esses arquivos, identificados pelo prefixo `"app.*"`, representam os componentes iniciais do sistema. Entre eles, o arquivo `app.component.html` define a estrutura visual principal da aplicação, enquanto o `app.component.scss` trata dos estilos globais. O arquivo `app.component.spec.ts` é responsável pelos testes automatizados do componente principal, e o `app.component.ts` implementa a lógica de funcionamento do mesmo. Complementando essa estrutura, o `app.config.ts` centraliza as configurações globais do Angular, e o `app.routes.ts` define as rotas da aplicação.

Agora, com relação aos diretórios, inicia-se uma organização macro que reflete a estrutura da arquitetura adotada, composta pelos seguintes módulos principais:

1. `core`: responsável pelas funcionalidades centrais utilizadas por todo o sistema, como configuração de serviços, instanciamento de classes globais, interceptadores, guardas de rotas, entre outras funcionalidades essenciais, mas que não estão vinculadas a nenhuma *feature* específica.

2. `features`: módulo no qual são declarados todos os submódulos do sistema, organizando as funcionalidades conforme suas respectivas responsabilidades, como *login*, registro, *home* e recuperação de senha.

3. `shared`: contém os componentes de software reutilizáveis em todo o código, como elementos de interface (campos de texto pré-configurados, *header* e *footer* padrões do sistema, etc.), entidades compartilhadas, utilitários, serviços, entre outros.

4. `styles`: relacionado aos estilos e suas configurações. Especificamente neste projeto, é nesse diretório que se encontra a configuração do Angular Material, conforme mencionado anteriormente.

Agora, iremos dar uma atenção especial ao módulo de `features`, que é, basicamente, onde está localizado o código das funcionalidades principais do sistema e onde aplicamos com maior rigor os princípios da Arquitetura Limpa. Cada funcionalidade foi cuidadosamente dividida em camadas bem definidas, respeitando a separação de responsabilidades e promovendo a manutenibilidade do sistema a longo prazo.

A Figura 3 apresenta uma visão geral da arquitetura adotada na camada de *Front-End*, evidenciando como as camadas `presentation`, `domain` e `data` se relacionam entre si e com o *Back-End*. Pode-se observar o fluxo de dados e responsabilidades entre os arquivos componentes da interface, as entidades e casos de uso do domínio, e os modelos, mapeadores e repositórios responsáveis pela comunicação com serviços externos. Essa estrutura modular reforça a coesão interna dos módulos e favorece a testabilidade, reutilização e escalabilidade do sistema.

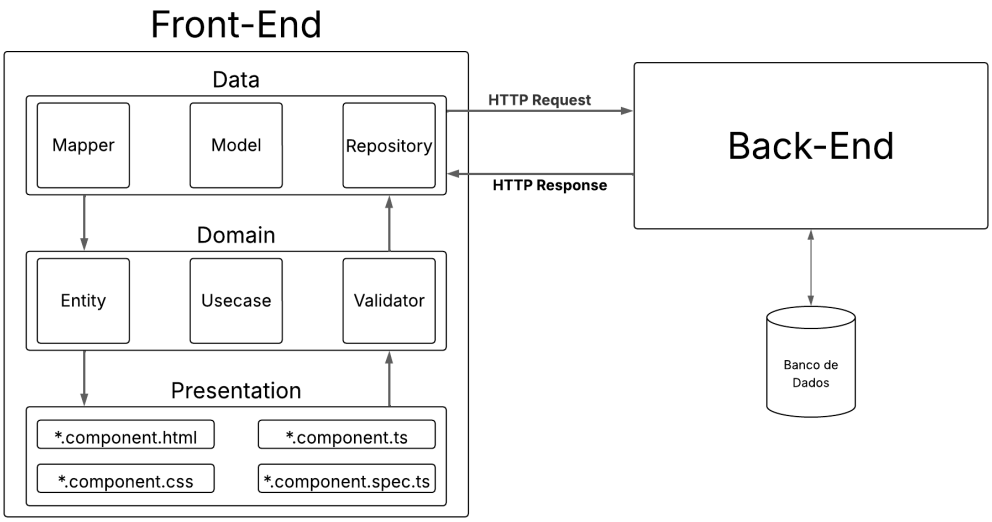


Figura 3 – Modelo aplicado da arquitetura limpa no sistema

Dessa forma, não iremos abordar detalhadamente cada funcionalidade, mas utilizaremos uma delas como exemplo, de modo que seja possível compreender como todas foram estruturadas na prática. Para fins de exemplificação da estrutura adotada, toma-se como referência o módulo de registro (`register`), mostrado na Figura 4.

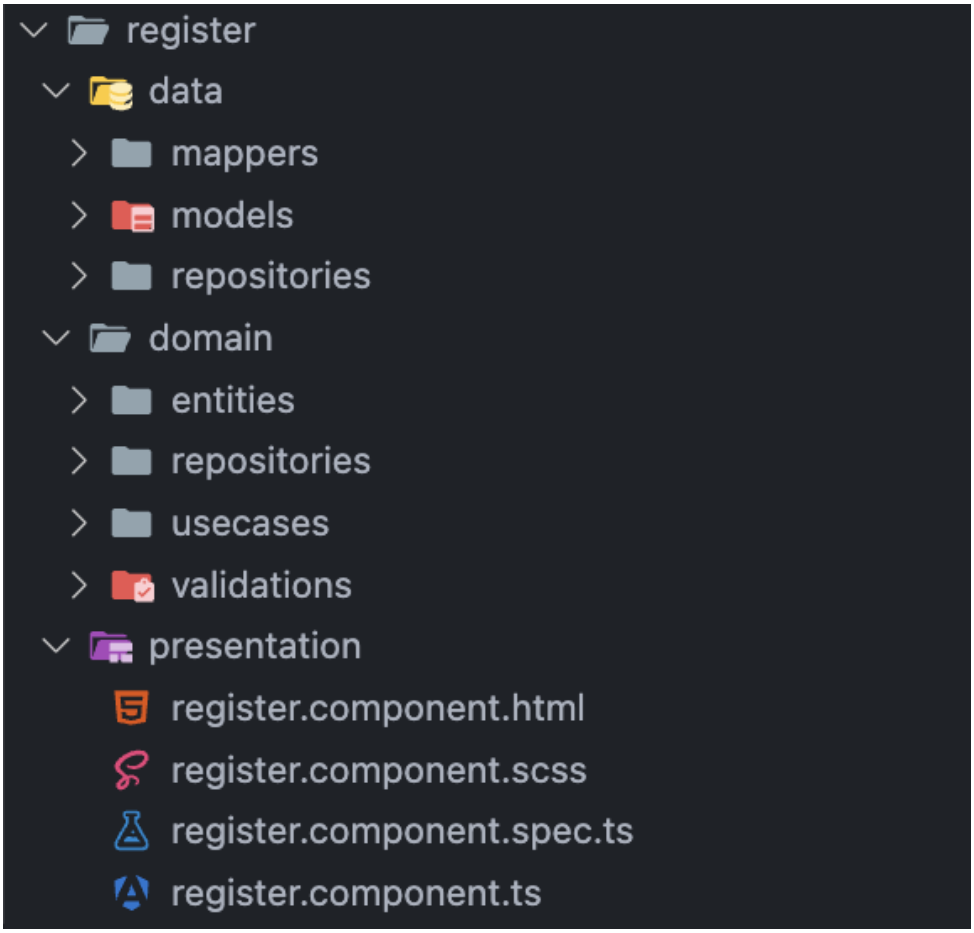


Figura 4 – Diretório `register`

Em sua estrutura interna, o módulo de registro adota um padrão arquitetural replicado em todos os demais módulos e submódulos funcionais do sistema. Essa organização baseia-se em três diretórios principais, que segmentam as responsabilidades entre a lógica de domínio, a comunicação externa e a interface com o usuário:

1. `data`: concentra os elementos relacionados às conexões externas do sistema, como as implementações responsáveis pelo consumo de APIs, os repositórios concretos e os mapeadores de dados. Esses mapeadores promovem a conversão entre os dados externos e as estruturas internas da aplicação, facilitando a integração entre camadas. Todas as implementações seguem contratos definidos no módulo `domain`, detalhado a seguir.

2. `domain`: representa o núcleo lógico de cada módulo, abrigando os conceitos fundamentais da aplicação, como as entidades de domínio, as interfaces dos repositórios e os casos de uso (`usecases`) que encapsulam os fluxos operacionais do sistema. Também estão presentes mecanismos de validação, como os arquivos auxiliares de verificação de formulários, e regras de negócio específicas de cada funcionalidade.

3. `presentation`: é a camada responsável pela interface com o usuário. Nela estão os arquivos próprios do Angular, como os templates visuais (`.html`), folhas de estilo (`.scss`), scripts de comportamento (`.ts`) e arquivos de testes (`.spec.ts`). O diretório também pode conter um subdiretório `components`, que agrupa componentes reutilizáveis, promovendo modularização e maior legibilidade do código.

Essa estrutura visa à separação clara de responsabilidades e à modularidade do sistema, promovendo maior facilidade de manutenção, escalabilidade e testabilidade. Entre os elementos centrais desse padrão arquitetural, destacam-se os conceitos de `Model`, `Entity` e `Mapper`, responsáveis por garantir o fluxo adequado de dados entre o mundo externo e a lógica interna da aplicação.

- **`Model`**: representa a estrutura dos dados recebidos de serviços externos, geralmente formatados em JavaScript Object Notation (JSON). São utilizados na camada `data` e refletem diretamente a resposta das APIs.
- **`Entity`**: define a estrutura de dados adotada internamente pela aplicação. As `entities` encapsulam regras de negócio, validações e funcionalidades específicas do domínio, e são empregadas na camada `domain`.
- **`Mapper`**: atua como intermediário entre `models` e `entities`, realizando a conversão entre os dois formatos. Dessa forma, garante-se que a lógica de negócio permaneça desacoplada da estrutura dos dados externos.

A seguir, apresenta-se a modelagem aplicada ao recurso **`AccessRestriction`** (restrições de acesso), com o objetivo de demonstrar como esses três elementos atuam em conjunto para manter o sistema coeso, modular e de fácil manutenção.

4.3.1.1 *Model*: `AccessRestrictionModel`

A classe `AccessRestrictionModel`, mostrada na Figura 5, define os campos recebidos da API no formato JSON. Ela contém os atributos básicos da entidade, com um método `fromJSON()` que organiza os dados conforme o esperado pelo sistema.

4.3.1.2 *Entity*: `AccessRestrictionEntity`

A classe `AccessRestrictionEntity`, mostrada na Figura 6, representa a entidade manipulada internamente nos casos de uso do sistema. Além de armazenar os mesmos dados do *model*, pode incorporar regras de negócio e métodos utilitários, como a formatação de datas por meio do método `createdAtFormatted()`.

```typescript
import { CreateAccessRestrictionsModel } from "./create-access-restrictions.model";

export class AccessRestrictionsModel extends CreateAccessRestrictionsModel {
  id: number;
  createdAt: string;

  constructor({
    id,
    reason,
    place,
    availableTime,
    device,
    connectionSpeed,
    screenResolution,
    classPresentationFormat,
    createdAt,
  }: {
    id: number,
    reason: string,
    place: string,
    availableTime: string,
    device: string,
    connectionSpeed: string,
    screenResolution: string,
    classPresentationFormat: string,
    createdAt: string,
  }) {
    super({
      reason,
      place,
      availableTime,
      device,
      connectionSpeed,
      screenResolution,
      classPresentationFormat,
    });
    this.id = id;
    this.createdAt = createdAt;
  }


  static fromJSON(data: any): AccessRestrictionsModel {
    return new AccessRestrictionsModel({
      id: data.id,
      reason: data.reason,
      place: data.place,
      availableTime: data.availableTime,
      device: data.device,
      connectionSpeed: data.connectionSpeed,
      screenResolution: data.screenResolution,
      classPresentationFormat: data.classPresentationFormat,
      createdAt: data.createdAt,
    });
  }
}
```

Figura 5 – Arquivo `access-restrictions.model.ts`

```typescript
import { CreateAccessRestrictionsEntity } from "./create-access-restrictions.entity";

export class AccessRestrictionsEntity extends CreateAccessRestrictionsEntity {
  id: number;
  createdAt: Date;

  constructor({
    id,
    reason,
    place,
    availableTime,
    device,
    connectionSpeed,
    screenResolution,
    classPresentationFormat,
    createdAt,
  }: {
    id: number,
    reason: string,
    place: string,
    availableTime: string,
    device: string,
    connectionSpeed: string,
    screenResolution: string,
    classPresentationFormat: string,
    createdAt: Date,
  }) {
    super({
      reason,
      place,
      availableTime,
      device,
      connectionSpeed,
      screenResolution,
      classPresentationFormat,
    });
    this.id = id;
    this.createdAt = createdAt;
  }

  get createdAtFormatted(): string {
    return `${this.createdAt.getDate()}/${this.createdAt.getMonth()}/${this.createdAt.getFullYear()}`;
  }
}
```

Figura 6 – Arquivo `access-restriction-entity.png`

4.3.1.3 *Mapper*: `AccessRestrictionMapper`

Por fim, o `AccessRestrictionMapper`, mostrado na Figura 7, realiza a conversão entre *model* e *entity*, herdando de uma classe genérica chamada `Mapper`. Essa herança padroniza o processo de transformação de dados, facilitando sua reutilização em outros módulos do sistema.

Essa abordagem modular contribui para a clareza do código, facilita a realização de testes unitários e promove maior escalabilidade do sistema, ao permitir que as diferentes camadas evoluam de forma independente.

4.3.2 Armazenamento Local

Dentre os serviços compartilhados, localizados no diretório `shared`, destaca-se um componente de software amplamente utilizado no contexto de desenvolvimento *Front-End*: o armazenamento local de dados. Esse recurso refere-se à persistência de informações

```typescript
import { Mapper } from "../../../../../core/base/mapper";
import { AccessRestrictionsEntity } from "../../domain/entities/access-restrictions.entity";
import { AccessRestrictionsModel } from "../models/access-restrictions.model";

export class AccessRestrictionsMapper extends Mapper<AccessRestrictionsEntity, AccessRestrictionsModel> {
  override mapFrom(entity: AccessRestrictionsEntity): AccessRestrictionsModel {
    return new AccessRestrictionsModel({
      id: entity.id,
      reason: entity.reason,
      place: entity.place,
      availableTime: entity.availableTime,
      device: entity.device,
      connectionSpeed: entity.connectionSpeed,
      screenResolution: entity.screenResolution,
      classPresentationFormat: entity.classPresentationFormat,
      createdAt: entity.createdAt.toISOString(),
    });
  }

  override mapTo(model: AccessRestrictionsModel): AccessRestrictionsEntity {
    return new AccessRestrictionsEntity({
      id: model.id,
      reason: model.reason,
      place: model.place,
      availableTime: model.availableTime,
      device: model.device,
      connectionSpeed: model.connectionSpeed,
      screenResolution: model.screenResolution,
      classPresentationFormat: model.classPresentationFormat,
      createdAt: new Date(model.createdAt),
    });
  }
}

```

Figura 7 – Arquivo `access-restriction-mapper.png`

diretamente no dispositivo do usuário, ou seja, do lado do cliente. Para isso, são utilizadas APIs nativas do navegador ou do sistema operacional, que possibilitam a leitura e gravação de dados de forma eficiente e segura.

Com base nesse conceito, foi desenvolvida uma interface genérica denominada `StorageBase`, responsável por definir os métodos que poderão ser utilizados para operações de armazenamento ao longo do sistema. A Figura 8 apresenta a estrutura da interface implementada.

A partir da interface `StorageBase`, foram desenvolvidas duas implementações com finalidades distintas. A primeira, baseada no *Local Storage*, tem como objetivo o uso genérico e irrestrito dentro da aplicação. Já a segunda, fundamentada no *Session Storage*, foi criada especificamente para armazenar dados relacionados à sessão do usuário.

Essa distinção entre as duas abordagens é relevante, uma vez que o *Local Storage* permite a persistência dos dados mesmo após o encerramento da aba ou navegador, mantendo as informações armazenadas até que o usuário as remova manualmente — seja limpando o *cache* ou os dados do site. Por outro lado, o *Session Storage* retém os dados somente durante a sessão ativa do navegador, sendo automaticamente descartados ao se fechar a aba ou janela correspondente.

Essa separação de responsabilidades possibilita uma gestão mais clara e eficiente dos dados que precisam ou não ser preservados ao longo de múltiplas execuções da apli-

```typescript
export abstract class StorageBase {
  protected storage!: Storage;

  protected constructor() { }

  setItem(key: string, value: unknown): void {
    if (typeof value == 'string') {
      this.storage.setItem(key, value);
    } else {
      this.storage.setItem(key, JSON.stringify(value));
    }
  }

  getItem(key: string): string | null {
    return this.storage.getItem(key);
  }

  getParsedItem<T = unknown>(key: string): T | null {
    const data = this.getItem(key);

    if (data) return JSON.parse(data) as T;

    return null;
  }

  removeItem(key: string): void {
    this.storage.removeItem(key);
  }

  hasItem(key: string): boolean {
    return this.storage.hasOwnProperty(key);
  }

  clear(): void {
    this.storage.clear();
  }
}
```

Figura 8 – Classe `StorageBase`

cação. A Figura 9 apresenta a implementação de ambas as abordagens, nas quais cada uma estende a interface `StorageBase` e utiliza, respectivamente, as instâncias nativas de `SessionStorage` e `LocalStorage`.

Para exemplificar seus usos dentro da aplicação, é possível destacar o serviço `AuthService`, mostrado na Figura 10, responsável por centralizar a autenticação do usuário (cujo funcionamento será explicado na próxima seção), e o `TranslationService`, mostrado na Figura 14, que gerencia as funcionalidades de internacionalização do sistema (as quais serão abordadas na seção seguinte).

session-storage.service.ts

src > app > shared > services > storage > session-storage > session-storage.service.ts > ...

```
You, 4 months ago | 1 author (You)
import { Injectable } from '@angular/core';
import { StorageBase } from '../storage-base';

You, 4 months ago | 1 author (You)
@Injectable({
  providedIn: 'root',
})
export class SessionStorageService extends StorageBase {
  override storage = sessionStorage;

  constructor() {
    super();
  }
}
```

local-storage.service.ts

src > app > shared > services > storage > local-storage > local-storage.service.ts > ...

```
You, 4 months ago | 1 author (You)
import { Injectable } from '@angular/core';
import { StorageBase } from '../storage-base';

You, 4 months ago | 1 author (You)
@Injectable({
  providedIn: 'root',
})
export class LocalStorageService extends StorageBase {
  override storage = localStorage;

  constructor() {
    super();
  }
}
```

Figura 9 – Classe `SessionStorageService` (à esquerda) e classe `LocalStorageService` (à direita)

```
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private sessionStorage = inject(SessionStorageService);
  private router = inject(Router);
  private sessionCredentials: Credentials | null = null;
```

Figura 10 – Classe `AuthService`

```
@Injectable({
  providedIn: 'root'
})
export class TranslationService {
  private readonly _translateService = inject(TranslateService);
  private readonly _localStorageService = inject(LocalStorageService);
```

Figura 11 – Classe `TranslationService`

4.3.3 Autenticação do Usuário

Em sistemas de informação, é essencial garantir a segurança tanto para o usuário quanto para o próprio sistema. Uma das formas mais comuns de se alcançar esse objetivo é por meio da autenticação do usuário nas requisições realizadas do cliente para o servidor. Assim, apenas os usuários devidamente autenticados conseguem acessar recursos protegidos, impedindo que chamadas não autorizadas sejam processadas. De modo geral, qualquer recurso que exija autenticação não estará acessível para usuários que não tenham iniciado uma sessão válida.

No contexto da plataforma CX, implementou-se esse mecanismo de segurança por meio do uso de *JSON Web Token (JWT)*. Ao realizar o processo de autenticação, o usuário recebe um *token*, que permanece válido durante a sessão ativa. Quando o *token* expira, o sistema redireciona o usuário automaticamente para a tela de *login*, exigindo

uma nova autenticação. Conforme mencionado na seção anterior, esse *token* é armazenado localmente por meio do serviço `AuthService`, que utiliza o `SessionStorageService` para garantir a persistência temporária dos dados de sessão.

A Figura 12 apresenta a implementação do serviço de autenticação do usuário, denominado `AuthService`, responsável por gerenciar o estado de autenticação, armazenar o *token* JWT e disponibilizar métodos auxiliares para sua recuperação e remoção:

```typescript
import { inject, Injectable } from '@angular/core';
import { CanActivateFn, Router } from '@angular/router';
import { jwtDecode } from 'jwt-decode';
import { SessionStorageService } from '../../../shared/services/storage';
import { AUTH_CREDENTIALS_KEY } from '../../../shared/services/storage/session-storage/session-storage-keys';
import { Credentials } from '../interfaces/auth.interface';

You, 3 months ago | 1 author (You)
interface JwtPayload {
  id: string;
}

You, 3 months ago | 1 author (You)
@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private sessionStorage = inject(SessionStorageService);
  private router = inject(Router);
  private sessionCredentials: Credentials | null = null;

  constructor() { }

  get credentials(): Credentials | null {
    return this.sessionCredentials || this.sessionStorage.getParsedItem<Credentials>(AUTH_CREDENTIALS_KEY);
  }

  set credentials(value: Credentials | null) {
    this.sessionCredentials = value;

    if (value) {
      this.sessionStorage.setItem(AUTH_CREDENTIALS_KEY, value);
    } else {
      this.sessionStorage.removeItem(AUTH_CREDENTIALS_KEY);
    }
  }

  get isAuthenticated(): boolean {
    return !!this.credentials?.accessToken;
  }

  set jwtCredentials(token: string) {
    const { id } = jwtDecode(token) as JwtPayload;

    this.credentials = {
      accessToken: token,
      id,
    };
  }

  logout() {
    this.credentials = null;
    this.router.navigateByUrl('/login');
  }

  canActivate(): boolean {
    if (!this.isAuthenticated) {
      this.router.navigateByUrl('/login');
    }

    return this.isAuthenticated;
  }
}

export const authGuard: CanActivateFn = () => {
  return inject(AuthService).canActivate();
};
```

Figura 12 – Classe `AuthService`

Resumidamente, esse serviço representa uma instância global no sistema, responsável por centralizar todas as variáveis e métodos relacionados à autenticação do usuário. Dessa forma, sempre que for necessário manipular dados de autenticação, recorre-se a esse serviço único, o que facilita o entendimento do fluxo de autenticação e sua aplicação no sistema por parte dos desenvolvedores. A classe fornece métodos para atribuição e recuperação da variável `sessionCredentials`, que armazena os dados da autenticação atual, além de métodos auxiliares que utilizam essas credenciais como base para a execução de ações específicas.

Adicionalmente, o sistema conta com um *guard* — mecanismo de proteção de rotas — que utiliza o serviço `AuthService` para permitir ou restringir o acesso de usuários a determinadas rotas, com base em sua autenticação. Esse mecanismo está definido no arquivo `app.routes.ts`, onde o `authGuard` é aplicado a todas as rotas protegidas dentro do módulo `Home`.

Por fim, destaca-se que todas as requisições realizadas para o *Back-End* do CX, enquanto o usuário estiver autenticado, devem conter o *token* de autenticação correspondente. Para isso, foi implementado um *interceptor*, mostrado na Figura 13, responsável por interceptar e modificar as chamadas realizadas pelo cliente HTTP da aplicação, adicionando o *token* de autenticação de forma transparente ao cabeçalho da requisição.

Como o sistema utiliza o padrão JWT para autenticação, o *token* de acesso (`accessToken`) armazenado no serviço `AuthService` é incluído no cabeçalho das requisições HTTP por meio do campo `Authorization`, utilizando o formato `Bearer Token`. Esse processo é realizado de forma automática pelo *interceptor*.

Caso ocorra algum problema relacionado à autenticação — como a ausência ou expiração do *token* —, o próprio sistema executa o procedimento de *logout*, encerrando a sessão do usuário. Além disso, para requisições que incluam o parâmetro de busca `noAuth`, o interceptor desconsidera a adição do cabeçalho `Authorization`, uma vez que essas chamadas não requerem autenticação. Essa abordagem é particularmente útil para integrações com APIs de terceiros ou chamadas públicas que não exigem credenciais.

4.3.4 Internacionalização

Com o avanço da globalização e o consequente aumento do alcance dos sistemas de informação, tornou-se essencial que aplicações ofereçam suporte a diferentes idiomas. No caso da plataforma CX, os idiomas selecionados foram o Português e o Inglês, sendo a definição inicial determinada com base na linguagem padrão do dispositivo do usuário ou, quando existente, na preferência previamente salva.

Todo o conteúdo textual exibido ao usuário é adaptado a partir dessa escolha, abrangendo tanto os elementos estáticos — como títulos de páginas, botões e parágrafos

```typescript
import { HttpErrorResponse, HttpEvent, HttpHandlerFn, HttpInterceptorFn, HttpRequest, HttpStatusCode } from '@angular/common/http';
import { inject } from '@angular/core';
import { Observable } from 'rxjs/internal/Observable';
import { throwError } from 'rxjs/internal/observable/throwError';
import { catchError } from 'rxjs/internal/operators/catchError';
import { environment } from '../../../../environment/environment';
import { AuthService } from '../services/auth.service';

export const KEY_NO_AUTH = 'noAuth';

export const authInterceptor: HttpInterceptorFn = (
  req: HttpRequest<any>,
  next: HttpHandlerFn
): Observable<HttpEvent<any>> => {
  if (!req.url.includes(environment.API_URL)) {
    return next(req);
  }

  const authService = inject(AuthService);

  let clonedRequest = req;

  if (clonedRequest.params.has(KEY_NO_AUTH)) {
    clonedRequest = clonedRequest.clone({
      params: clonedRequest.params.delete(KEY_NO_AUTH),
    });
  } else if (authService.isAuthenticated) {
    clonedRequest = clonedRequest.clone({
      setHeaders: {
        Authorization: `Bearer ${authService.credentials?.accessToken}`,
      },
    });
  }

  return next(clonedRequest).pipe(catchError(error => errorHandler(error, authService)));
};

function errorHandler(response: HttpErrorResponse, authService: AuthService): Observable<HttpEvent<unknown>> {
  if (response.status === HttpStatusCode.Unauthorized) {
    authService.logout();
  }

  return throwError(() => response);
}
```

Figura 13 – *Auth Interceptor*

presentes em arquivos no formato JSON específicos para cada idioma — quanto os elementos dinâmicos, incluindo as respostas das requisições realizadas ao *Back-End*. Estas respostas são personalizadas com base no código do idioma informado pelo *Front-End*, como, por exemplo, `"pt-BR"` e `"en-US"`.

A implementação da lógica de internacionalização foi centralizada no serviço `TranslationService`, mostrado na Figura 14.

```typescript
import { isPlatformBrowser } from '@angular/common';
import { Inject, Injectable, PLATFORM_ID, inject } from '@angular/core';
import { MatIconRegistry } from '@angular/material/icon';
import { DomSanitizer } from '@angular/platform-browser';
import { TranslateService } from '@ngx-translate/core';
import { Observable } from 'rxjs/internal/Observable';
import { map } from 'rxjs/internal/operators/map';
import { LocalStorageService } from '../../../shared/services/storage';
import { languages, languagesLabels } from '../translation.languages';


@Injectable({
  providedIn: 'root',
})
export class TranslationService {
  private readonly _translateService = inject(TranslateService);
  private readonly _localStorageService = inject(LocalStorageService);

  defaultLang = 'pt-BR';

  get currentLang(): string {
    return this._translateService.currentLang;
  }

  get currentLangStream(): Observable<string> {
    return this._translateService.onLangChange.pipe(map(event => event.lang));
  }

  get currentLangLabel(): string {
    const langIndex = languages.indexOf(this.currentLang);
    return languagesLabels[langIndex];
  }

  getTranslatedValue(key: string): string {
    return this._translateService.instant(key);
  }

  constructor(@Inject(PLATFORM_ID) private platformId: Object) {
    if (isPlatformBrowser(this.platformId)) {
      const browserLang = navigator.language || navigator.languages[0];
      const savedLang = this._localStorageService.getItem('lng');

      this.defaultLang = savedLang ? savedLang : browserLang;
      this._translateService.setDefaultLang(this.defaultLang);
      this._translateService.use(this.defaultLang);
    }

    this.registerLanguagesSvgIcons();
  }

  changeLang(lang: string) {
    this._translateService.use(lang);
    if (isPlatformBrowser(this.platformId)) {
      this._localStorageService.setItem('lng', lang);
    }
  }

  private registerLanguagesSvgIcons() {
    const sanitizer = inject(DomSanitizer);
    const iconRegistry = inject(MatIconRegistry);

    for (const lang of languages) {
      const svgContent = 'assets/i18n/flags/' + lang + '.svg';
      iconRegistry.addSvgIcon(lang, sanitizer.bypassSecurityTrustResourceUrl(svgContent));
    }
  }
}
```

Figura 14 – Classe `TranslationService`

Assim como o `AuthService`, trata-se de uma instância global que centraliza todas as funcionalidades relacionadas à tradução no sistema. Sua variável principal, `defaultLang`, representa o idioma atualmente selecionado, sendo utilizada por diversos métodos internos. O serviço também injeta dinamicamente duas dependências: o `TranslateService`, uma biblioteca do Angular voltada à tradução de conteúdo textual, e o `LocalStorageService`, responsável por persistir localmente a escolha de idioma do usuário.

Como a internacionalização também influencia as comunicações com o *Back-End*, foi criado um *interceptor* dedicado à modificação de requisições HTTP, mostrado na Figura 15.

```
import { HttpInterceptorFn } from '@angular/common/http';
import { inject, Injector } from '@angular/core';
import { environment } from '../../../../environment/environment';
import { TranslationService } from '../services/translation.service';

export const translationInterceptor: HttpInterceptorFn = (req, next) => {
  if (!req.url.includes(environment.API_URL)) {
    return next(req);
  }

  const injector = inject(Injector);
  const translationService = injector.get(TranslationService);

  let clonedRequest = req;

  clonedRequest = clonedRequest.clone({
    setHeaders: {
      Language: translationService.currentLang,
    },
  });

  return next(clonedRequest);
};
```

Figura 15 – *Translation Interceptor*

Esse interceptor adiciona o cabeçalho `Language` em todas as chamadas feitas ao servidor, definindo-o com o valor atual de `currentLang`, gerenciado pelo `TranslationService`, independentemente de o usuário estar autenticado ou não.

4.3.5 Documentação do Sistema

Durante o processo de desenvolvimento de software, torna-se essencial a elaboração de registros e documentos que descrevam o funcionamento do sistema, abordando aspectos como arquitetura, padrões de codificação, estilização de componentes (no caso do *Front-End*), configuração do ambiente de desenvolvimento, processos de compilação (*build*) e execução da aplicação, entre outros. Essa prática contribui significativamente para a agilidade e a continuidade do projeto, especialmente quando há a necessidade de manutenção ou evolução por parte de outros desenvolvedores no futuro.

No presente trabalho, a documentação foi estruturada por meio de arquivos no formato Markdown (com extensão `".md"`), inseridos diretamente no repositório do projeto.

Dessa forma, os documentos também são versionados juntamente com o código-fonte, garantindo sua atualização contínua. Esses arquivos foram organizados dentro do diretório `documentation`, sendo referenciados no arquivo principal de apresentação do projeto, o `README`, mostrado na Figura 16.

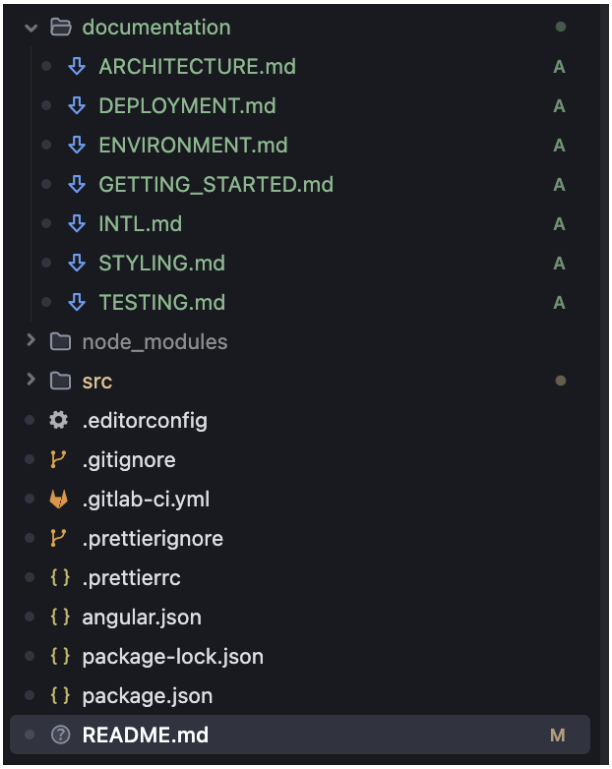


Figura 16 – Diretório raiz do projeto, destacando o diretório `documentation` e o arquivo `README`

Para exemplificar como foi feita a documentação do sistema, a Figura 17 apresenta o conteúdo do arquivo `README.md`, juntamente com um dos arquivos complementares presentes no diretório `documentation`.

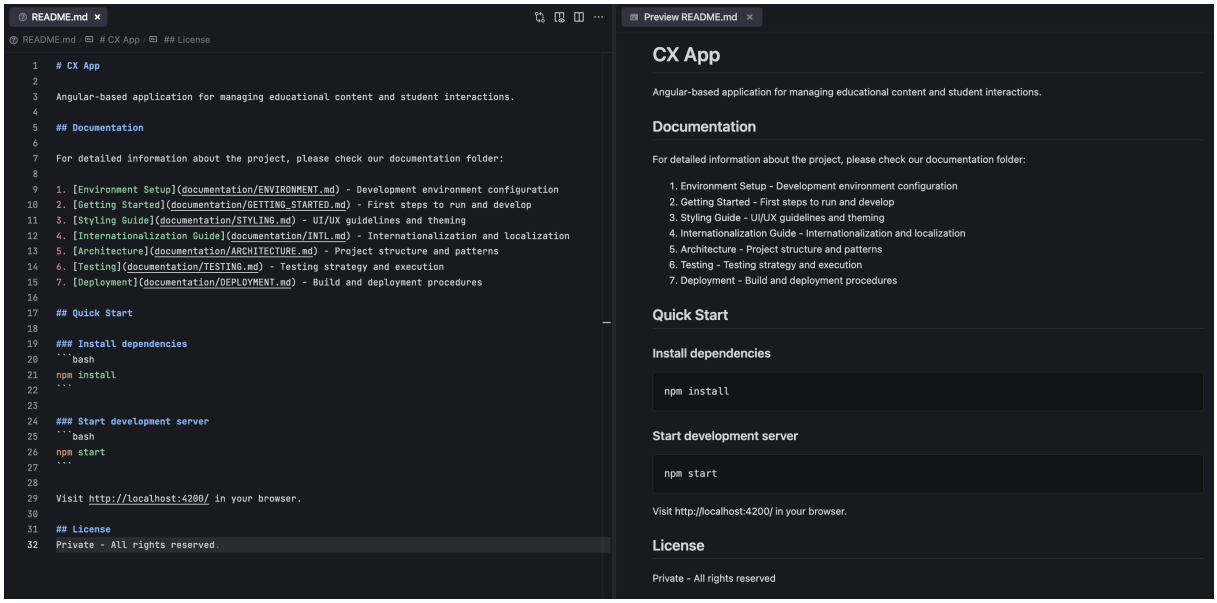


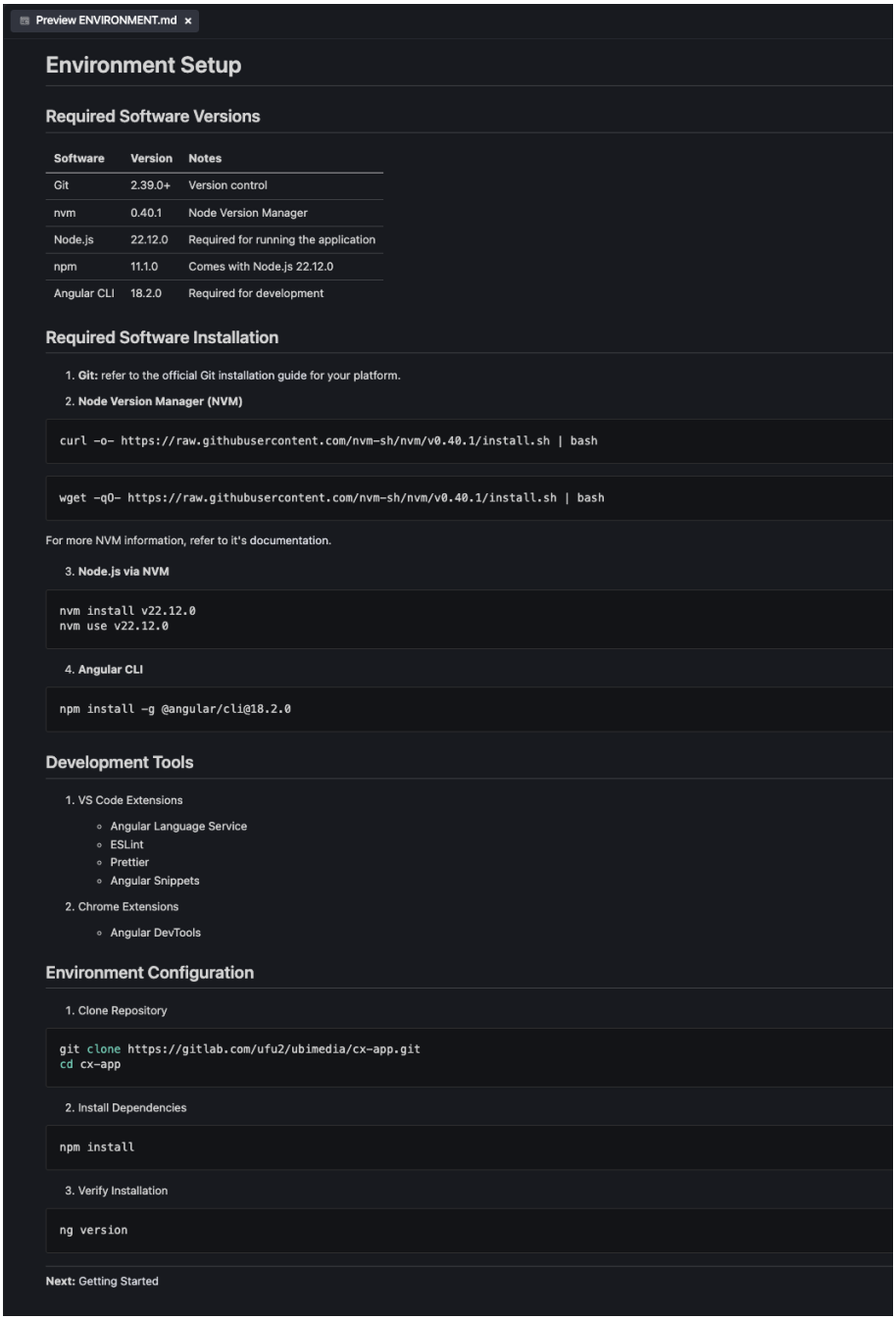
Figura 17 – `README.md`

O arquivo apresentado anteriormente serve como base para a documentação do repositório. Nele, é possível observar uma breve introdução ao projeto, uma seção dedicada à navegação pelas demais documentações e seus respectivos *links*, além de instruções para início rápido e informações sobre a licença do software. Dessa forma, concentram-se, de maneira sucinta, os principais dados sobre o sistema. Caso o leitor deseje informações adicionais, pode acessar os demais documentos que abordam, com maior profundidade, os tópicos relevantes.

No arquivo `ENVIRONMENT.MD`, mostrado na Figura 18, são listadas as versões dos softwares e *frameworks* utilizados, bem como instruções para sua instalação e configuração de forma rápida e simplificada. Quando necessário, são disponibilizados links para as documentações oficiais, permitindo que o desenvolvedor aprofunde seu conhecimento ou esclareça dúvidas.

4.3.6 Controle de Versão e Repositório de Código

Projetos de pequena a grande escala utilizam o controle de versão como uma estratégia fundamental para otimizar o desenvolvimento de software, garantindo maior segurança, precisão e rastreabilidade nas alterações realizadas ao longo do tempo. Esse mecanismo permite que equipes atuem de forma colaborativa e organizada, seguindo fluxos estruturados, como o GitFlow. O Git é um sistema de controle de versões distribuído, amplamente adotado no setor, que oferece a base para gerenciar o código-fonte de maneira eficiente. A partir dele, surgiram diversas plataformas especializadas, como GitLab, GitHub e BitBucket.



Preview ENVIRONMENT.md

Environment Setup

Required Software Versions

Software	Version	Notes
Git	2.39.0+	Version control
nvm	0.40.1	Node Version Manager
Node.js	22.12.0	Required for running the application
npm	11.1.0	Comes with Node.js 22.12.0
Angular CLI	18.2.0	Required for development

Required Software Installation

1. **Git:** refer to the official Git installation guide for your platform.
2. **Node Version Manager (NVM)**

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
```

```
wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
```

For more NVM information, refer to it's documentation.

3. **Node.js via NVM**

```
nvm install v22.12.0
nvm use v22.12.0
```

4. **Angular CLI**

```
npm install -g @angular/cli@18.2.0
```

Development Tools

1. VS Code Extensions
   - Angular Language Service
   - ESLint
   - Prettier
   - Angular Snippets
2. Chrome Extensions
   - Angular DevTools

Environment Configuration

1. Clone Repository

```
git clone https://gitlab.com/ufu2/ubimedia/cx-app.git
cd cx-app
```

2. Install Dependencies

```
npm install
```

3. Verify Installation

```
ng version
```

Next: Getting Started

Figura 18 – `ENVIRONMENT.md`

Durante o desenvolvimento deste projeto, foram adotados princípios do GitFlow para organizar e manter o repositório. Foi criada uma *branch* principal denominada *dev*, atualizada por meio de *Merge Request (MR)* sempre que uma nova funcionalidade, ajuste ou melhoria estivesse finalizada e pronta para testes. Para cada nova alteração, o desenvolvedor criava uma *branch* local a partir da *dev* e, após concluir o desenvolvimento, submetia-a ao repositório remoto por meio de um MR, o que permitia o controle adequado do fluxo de modificações e facilitava o processo de revisão.

Uma das vantagens oferecidas pelas plataformas baseadas em Git é a integração nativa com sistemas de Integração Contínua e Entrega Contínua (CI/CD). O GitLab, plataforma escolhida para este projeto, possui recursos integrados que permitem a definição de *jobs* — etapas automatizadas de execução de tarefas, como testes, compilação e implantação — e de *pipelines*, que organizam essas tarefas em fluxos contínuos e automatizados de entrega.

Mais detalhes sobre os conceitos de CI/CD e sua implementação neste projeto serão apresentados na seção seguinte.

4.3.7 Entrega e Integração Contínua (CI/CD)

A Entrega e Integração Contínua, conhecida pela sigla CI/CD, é uma prática fundamental da cultura Development, Security and Operations (DevSecOps). Essa abordagem visa otimizar o fluxo de trabalho entre as equipes de desenvolvimento, segurança e operações, promovendo uma entrega de software mais rápida, segura e confiável. Por meio de processos automatizados, o CI/CD facilita a criação, teste, implantação e manutenção contínua de aplicações, com ciclos de *feedback* mais curtos e eficazes.

Como um dos pilares do DevSecOps, o CI/CD permite a automação de diversas etapas do desenvolvimento e entrega de software. Suas *pipelines*, compostas por sequências de tarefas automatizadas, possibilitam desde a execução de testes unitários e de integração até testes de performance e a implantação contínua em diferentes ambientes. Essa automação reduz a incidência de erros humanos, aumenta a eficiência do time e proporciona maior previsibilidade nos ciclos de entrega.

A Figura 19 ilustra o arquivo responsável por definir a estrutura da *pipeline* no contexto deste projeto, utilizando a linguagem de marcação Yet Another Markup Language (YAML), amplamente empregada para a escrita de arquivos de configuração em ambientes DevOps.

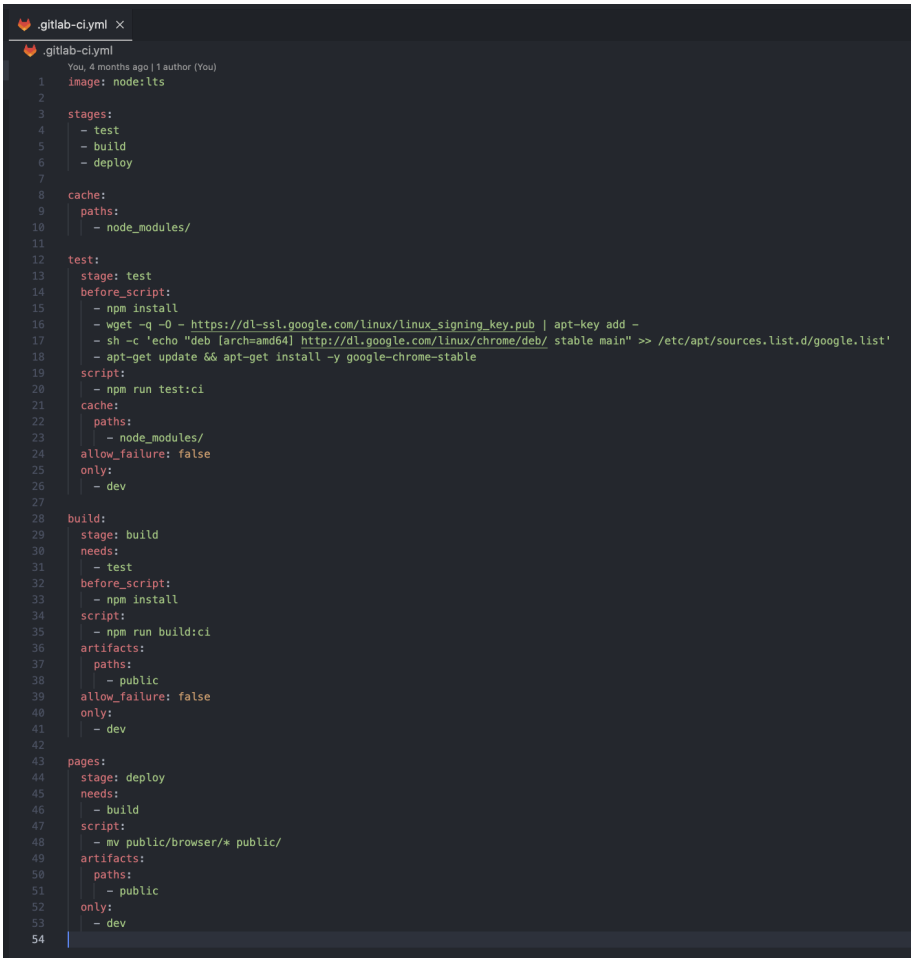


Figura 19 – Arquivo `.gitlab-ci.yml`

A seguir, apresentam-se as principais definições configuradas no arquivo de *pipeline*:

1. **`image`**: define qual imagem Docker será utilizada para executar os *jobs*, ou seja, o ambiente base para a execução dos comandos configurados;

2. **`stages`**: especifica as etapas principais da *pipeline*, que neste caso são três: `test`, `build` e `deploy`. Para cada uma dessas etapas, é possível associar diferentes *jobs*, que serão executados sequencialmente conforme a ordem definida;

3. **`cache`**: determina quais diretórios ou arquivos serão mantidos em cache (persistência local temporária) durante a execução da *pipeline*, com o intuito de acelerar as execuções subsequentes;

4. **`jobs`**: refere-se às tarefas principais declaradas no arquivo, neste caso: `test`, `build` e `pages`. Cada *job* descreve como deve ser executado dentro de uma etapa específica. Entre suas propriedades, destacam-se: **`needs`**, que define dependências entre *jobs*; `only`, que especifica em quais *branches* os *jobs* devem ser executados (por exemplo,

apenas na *branch* `dev`); e os blocos `before-script` e `script`, responsáveis por listar os comandos que serão de fato executados durante a execução do *job*.

Abaixo, apresentam-se exemplos da execução das *pipelines* diretamente na interface *Web* do GitLab, possibilitando o acompanhamento em tempo real de cada etapa, bem como a verificação de falhas ou êxitos ao longo do processo. Na Figura 20, observa-se uma execução bem-sucedida da *pipeline*, com todas as etapas finalizadas corretamente conforme o fluxo de integração contínua definido.

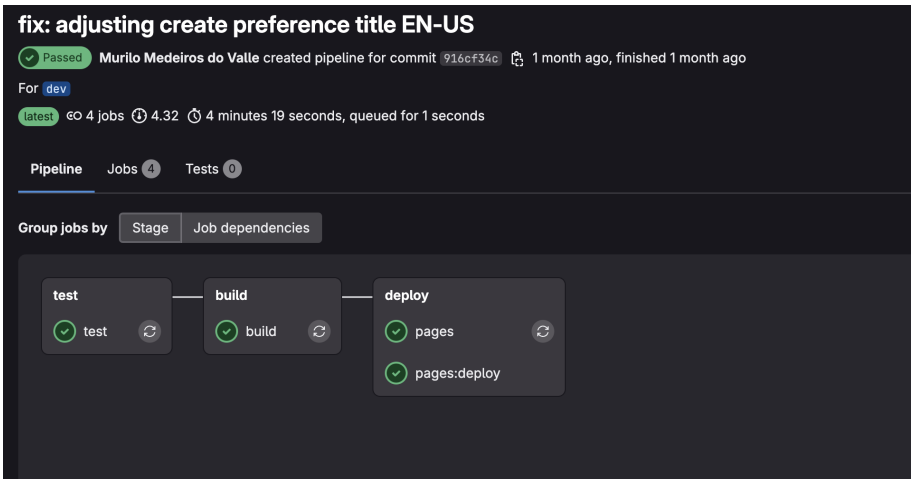


Figura 20 – Exemplo de *pipeline* executada com sucesso

A Figura 21 apresenta uma execução interrompida da *pipeline* em decorrência de falhas nos testes automatizados. Conforme configurado na regra de dependência do arquivo `.gitlab-ci.yml`, as etapas subsequentes são canceladas automaticamente, impedindo a continuidade da integração.

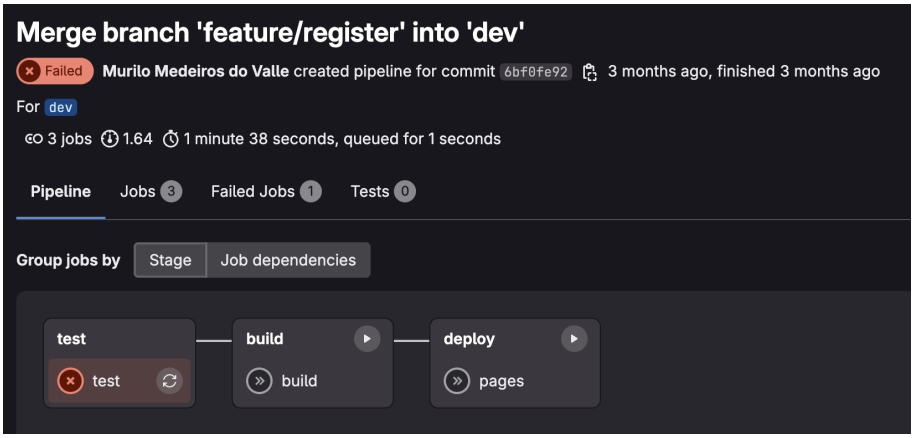


Figura 21 – Exemplo de *pipeline* com falha na etapa de testes

De modo semelhante, a Figura 22 contém um cenário de falha na etapa de `build`.

Como definido na lógica da *pipeline*, esse erro também impede a progressão para o *deploy*, garantindo que apenas códigos válidos e testados sigam para os ambientes seguintes.

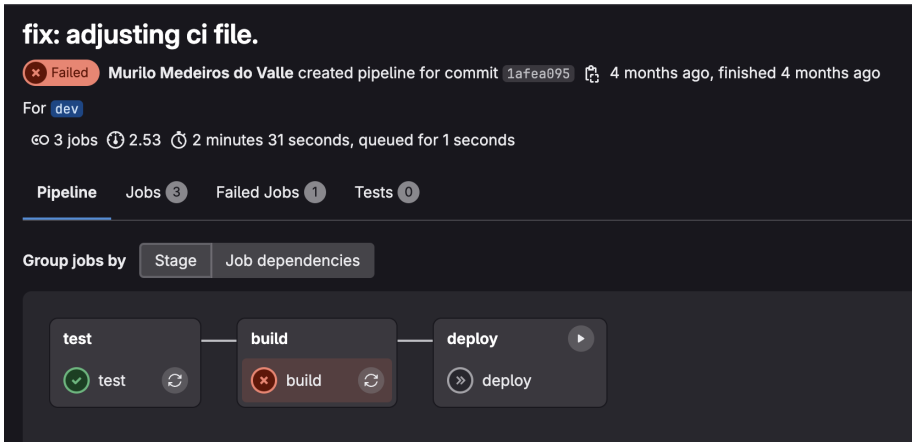


Figura 22 – Exemplo de *pipeline* com falha na etapa de `build`

5 Resultados

Esta seção apresenta os principais resultados obtidos a partir da modernização do sistema CX. A proposta de reconstrução da interface visou não apenas adequar o sistema às exigências técnicas e visuais mais atuais, mas também proporcionar melhorias tangíveis na experiência do usuário e na eficiência da aplicação. Os resultados aqui descritos contemplam, portanto, dois aspectos centrais: as melhorias visuais e de usabilidade promovidas pela nova interface e os ganhos técnicos obtidos em termos de desempenho e qualidade da aplicação, medidos por ferramentas especializadas.

5.1 Visuais e de Usabilidade

Com a implementação da nova interface, observou-se uma melhoria significativa na organização visual do sistema, na consistência dos estilos adotados e na clareza dos elementos interativos. A adoção de uma identidade visual mais limpa e atual, aliada às boas práticas de UX/UI, contribuiu para tornar a experiência do usuário mais acessível, intuitiva e coerente com padrões modernos de usabilidade.

A seguir, são apresentados comparativos visuais entre a versão anterior do sistema (identificada nas imagens como *[OLD]*) e a versão atualizada (identificada como *[NEW]*). Cabe destacar que, por se tratar de um sistema originalmente não responsivo, não foram incluídas imagens da versão antiga em dispositivos móveis, uma vez que tal comparação não se aplica ao escopo desta análise.

As Figuras 23 e 24 apresentam, respectivamente, as telas de login na versão anterior e na versão atualizada do sistema. É possível observar uma reorganização dos elementos, com melhorias na hierarquia visual, na clareza e na acessibilidade, proporcionando uma experiência mais agradável para o usuário.

Figura 23 – Página de *Login* [*OLD*]

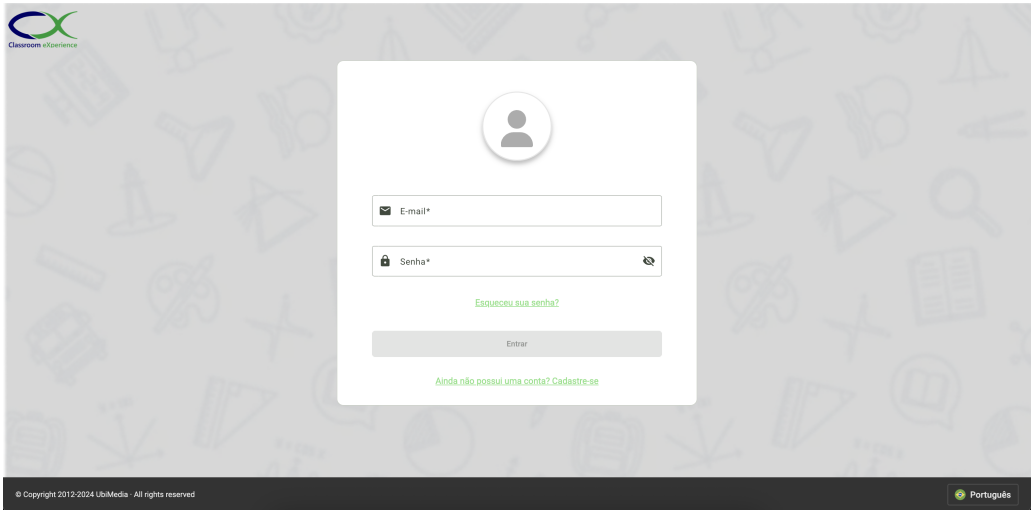


Figura 24 – Página de *Login* [*NEW*]

Na sequência, as Figuras 25 e 26 ilustram a tela de registro. A nova interface adota um layout mais simples e alinhado às práticas atuais de usabilidade, com foco na objetividade e na facilidade de preenchimento.

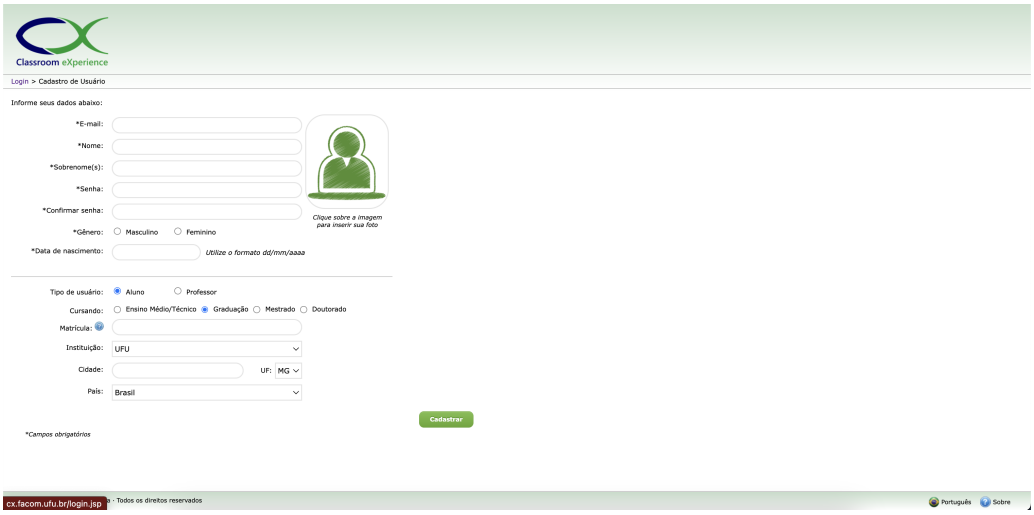


Figura 25 – Página de Registro [*OLD*]

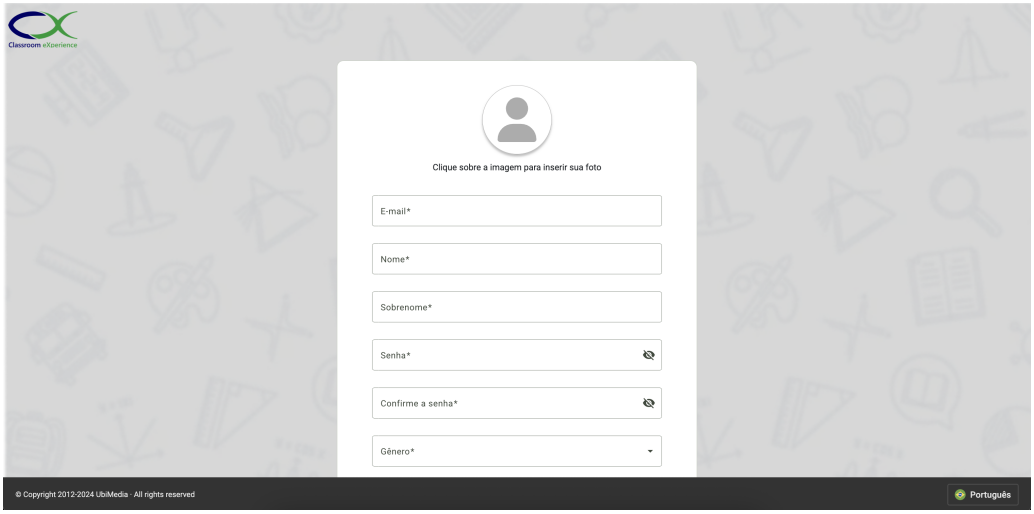


Figura 26 – Página de Registro [*NEW*]

As Figuras 27 e 28 comparam a tela de disciplinas. A atualização trouxe uma organização mais intuitiva das informações, facilitando a navegação entre os conteúdos. Além disso, a inclusão de imagens ilustrativas auxilia na identificação visual dos cursos, contribuindo para uma interação mais dinâmica.

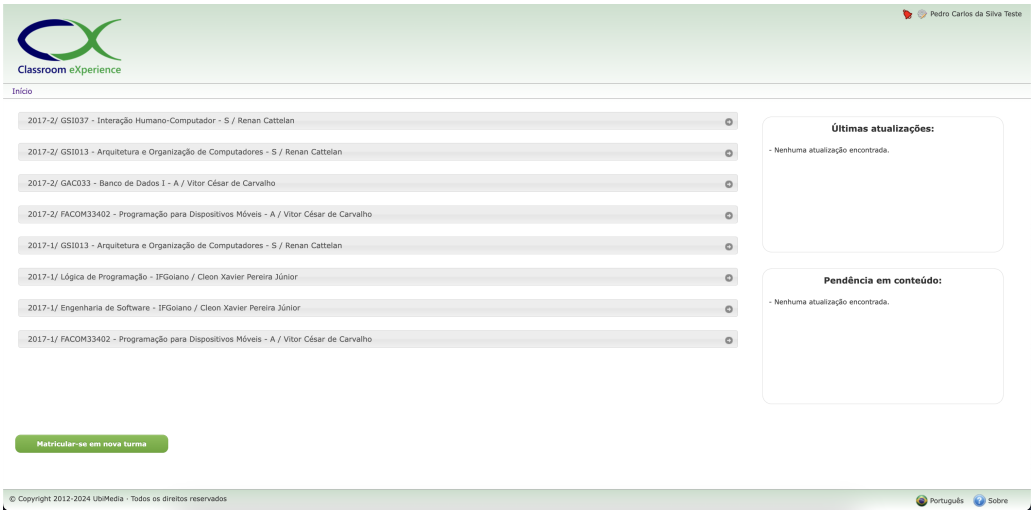


Figura 27 – Página de Disciplinas [*OLD*]

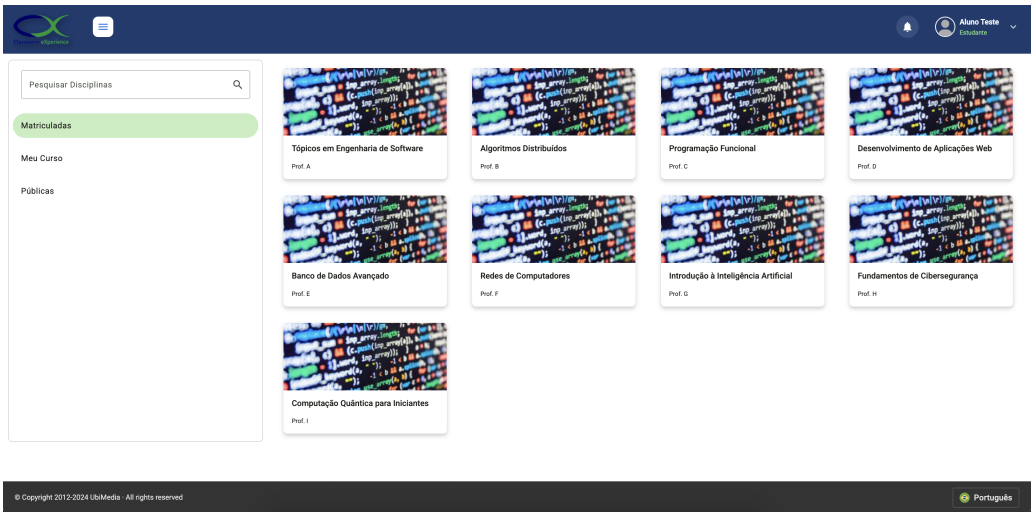


Figura 28 – Página de Disciplinas [*NEW*]

Dando continuidade, as Figuras 29 a 31 representam a página da disciplina. Enquanto a versão anterior apresentava os dados de forma mais segmentada, o novo layout oferece uma visualização mais clara, integrando os principais elementos de maneira estruturada e objetiva.

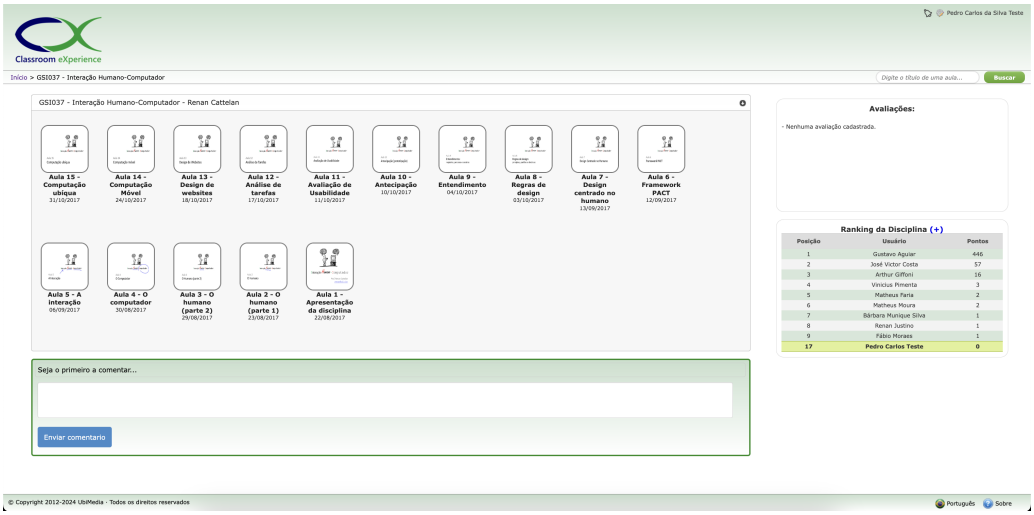


Figura 29 – Página da Disciplina [*OLD*]

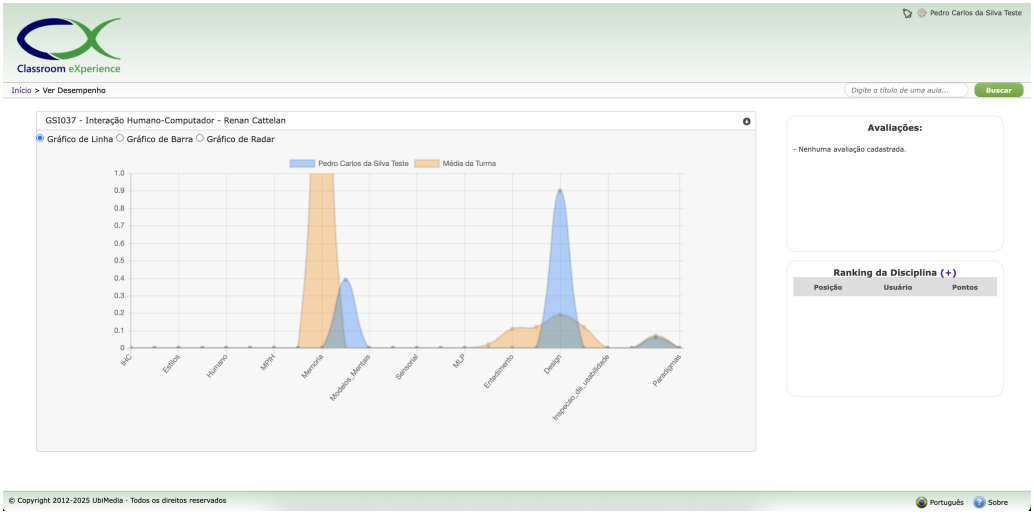


Figura 30 – Página de Desempenho na Disciplina [*OLD*]

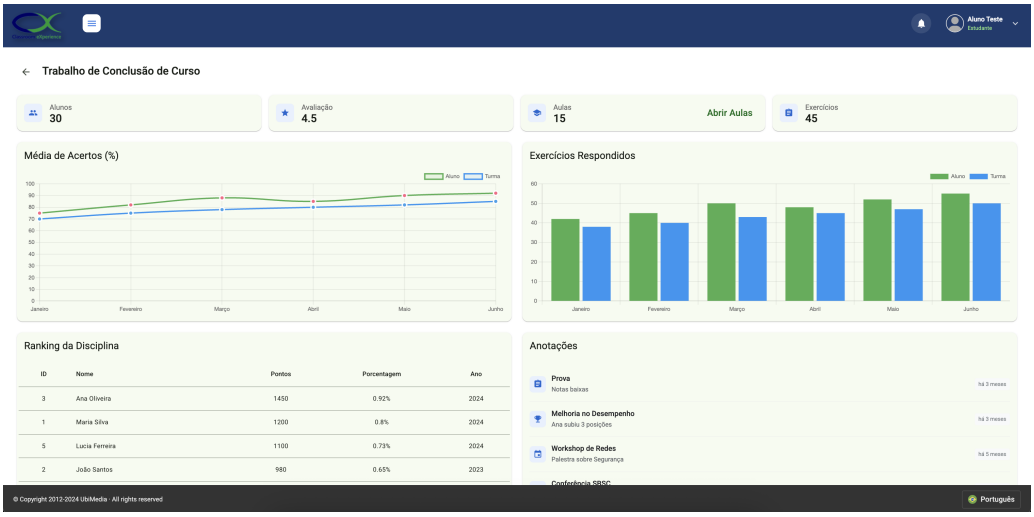


Figura 31 – Página da Disciplina [*NEW*]

Complementando essa visualização, a Figura 32 apresenta a nova tela dedicada às aulas da disciplina, que oferece uma navegação direta e organizada entre os conteúdos disponibilizados.

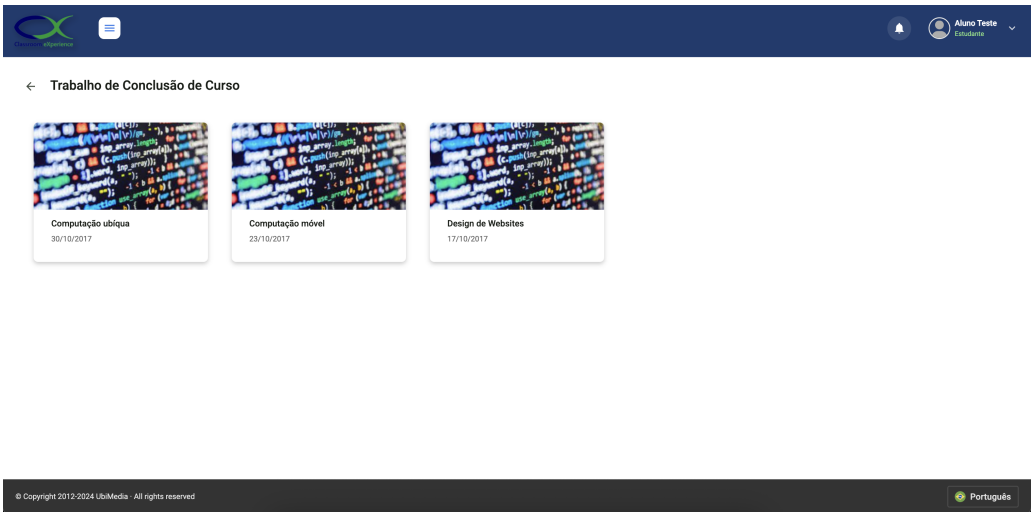


Figura 32 – Página das Aulas da Disciplina [*NEW*]

As Figuras 33 e 34 demonstram a tela de dados pessoais. A reformulação proporcionou uma disposição mais clara dos campos e informações, agora incorporados a uma estrutura unificada denominada “Configurações da Conta”. Este agrupamento, disposto em abas na lateral esquerda, reúne funcionalidades como Dados Pessoais, Contexto de Acesso, Preferências, entre outras, promovendo uma navegação mais fluida e centralizada.

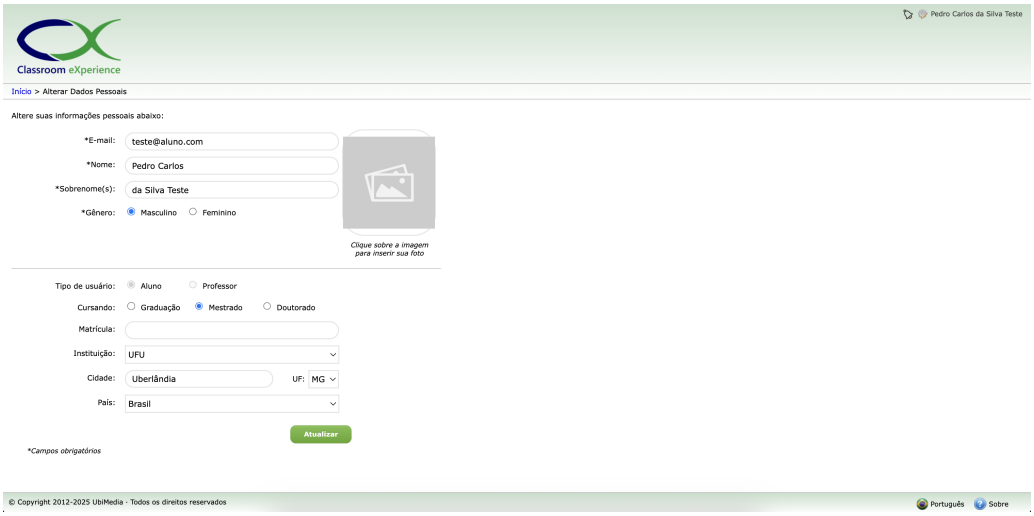


Figura 33 – Página de Dados Pessoais [*OLD*]

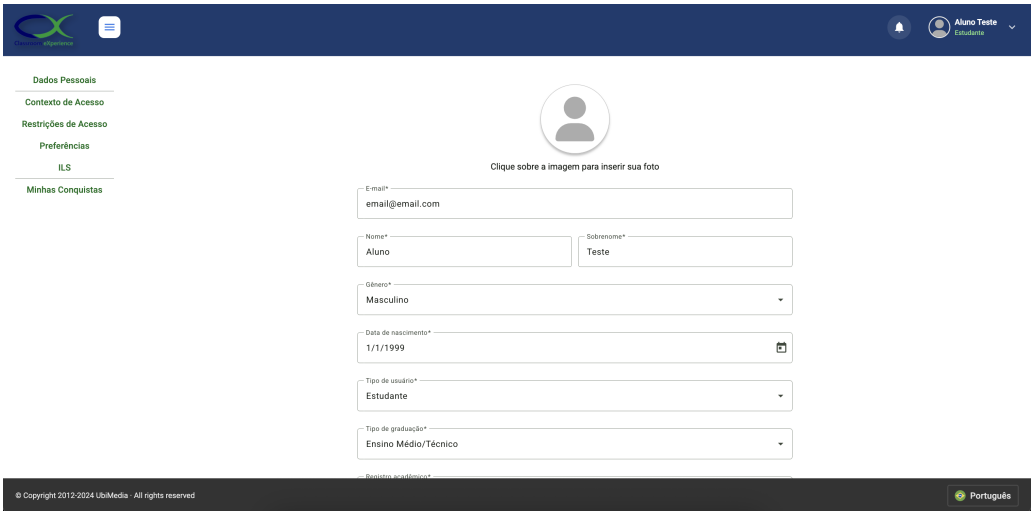


Figura 34 – Página de Dados Pessoais [*NEW*]

De forma alinhada, as Figuras 35 e 36 ilustram a tela de Contexto de Acesso, que passou a integrar esse mesmo agrupamento dentro do menu de configurações. A nova interface apresenta uma organização mais limpa e objetiva, contribuindo para uma experiência de uso mais prática.

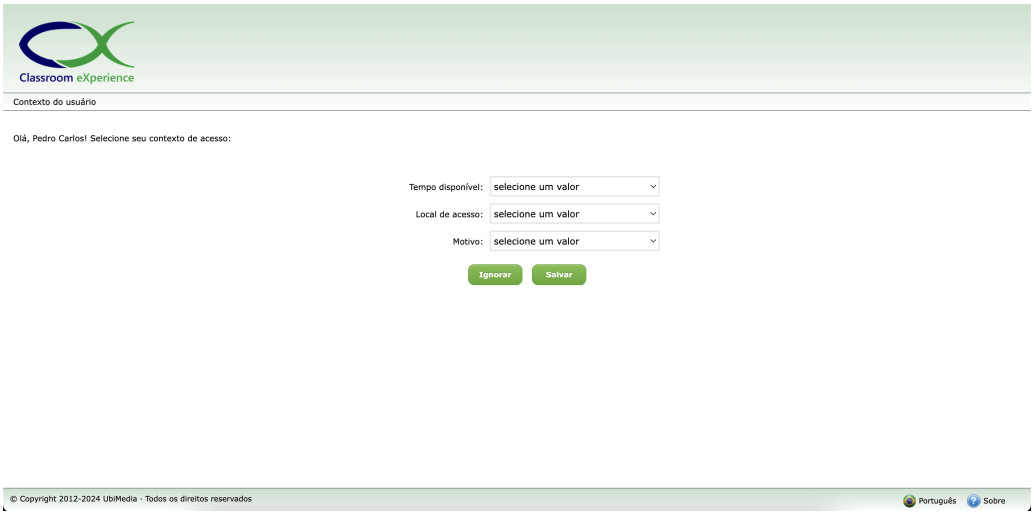


Figura 35 – Página de Contexto de Acesso [*OLD*]

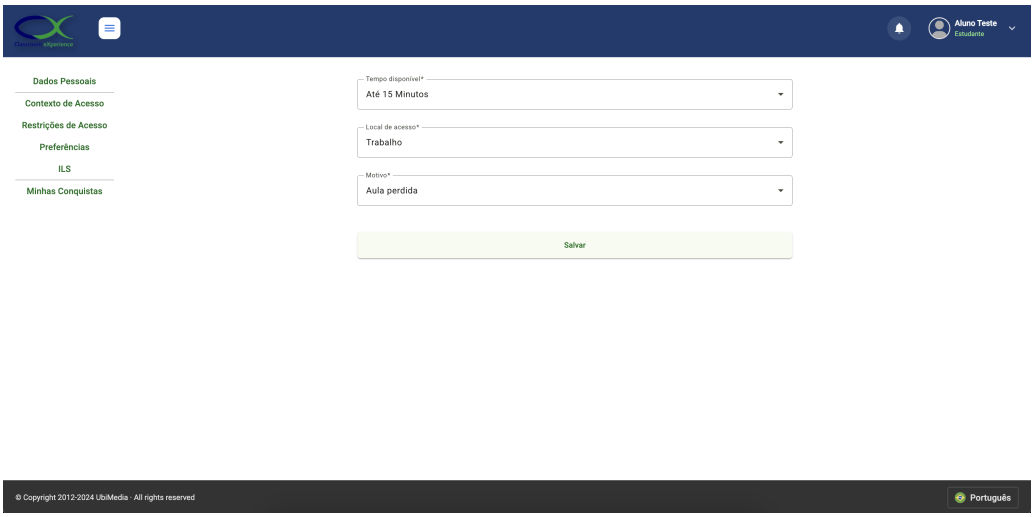


Figura 36 – Página de Contexto de Acesso [*NEW*]

As Figuras 37 e 38 comparam a tela de Restrições de Acesso. A atualização trouxe uma interface mais moderna, com melhor aproveitamento do espaço e apresentação dos dados de forma mais clara, o que torna a interação mais intuitiva.

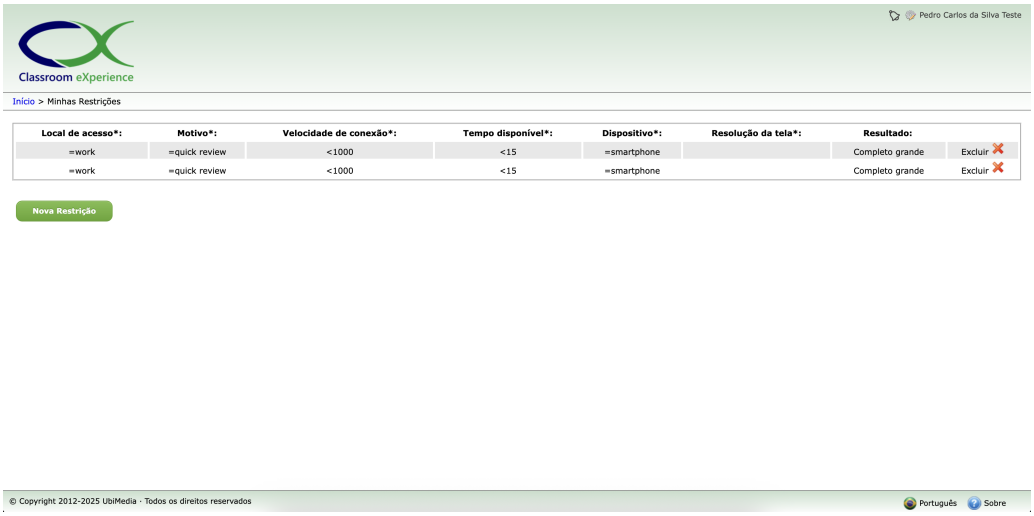


Figura 37 – Página de Restrições de Acesso [*OLD*]

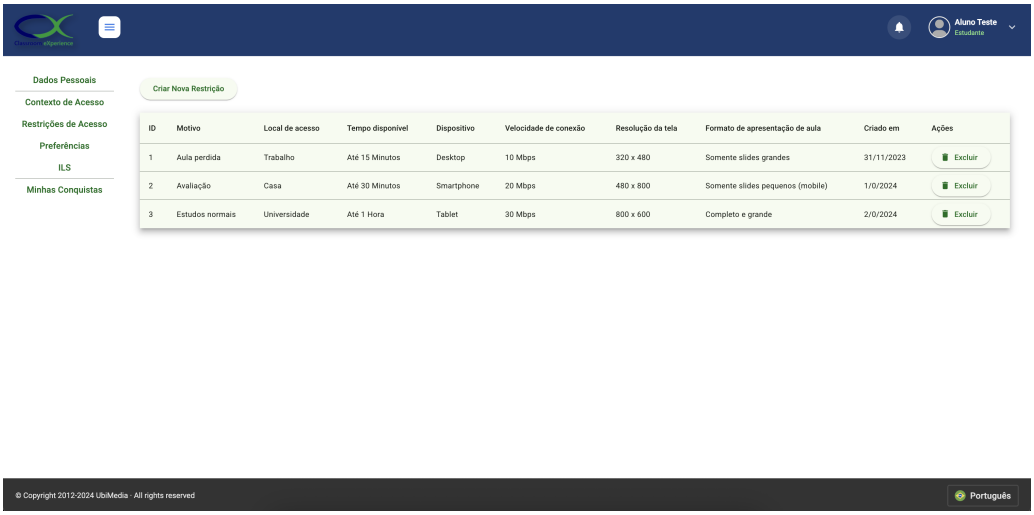


Figura 38 – Página de Restrições de Acesso [*NEW*]

Esse mesmo padrão de melhorias pode ser observado nas Figuras 39 e 40, que mostram a tela de Criação de Restrição de Acesso. O novo layout contribui para um processo de preenchimento mais simples e organizado, minimizando erros e facilitando a compreensão dos campos necessários.

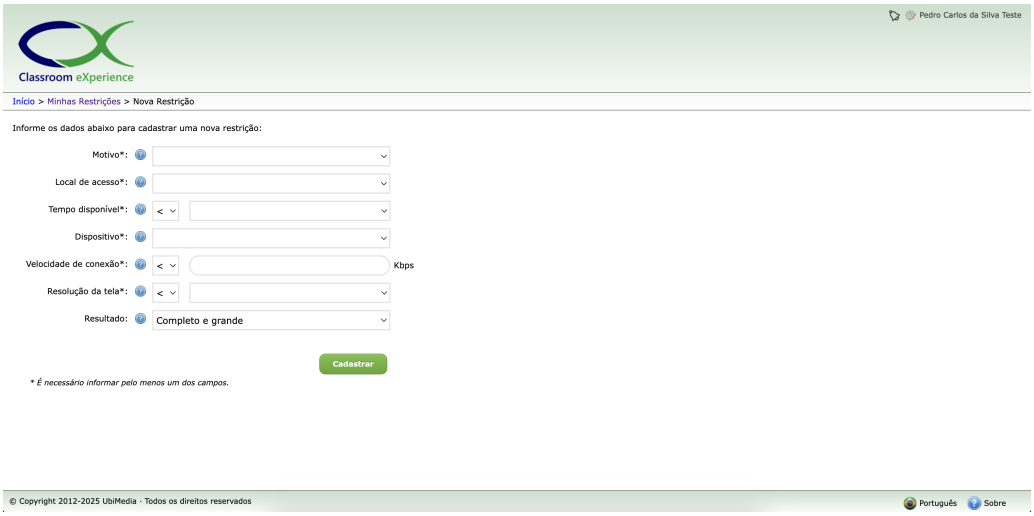


Figura 39 – Página de Criação de Restrição de Acesso [*OLD*]

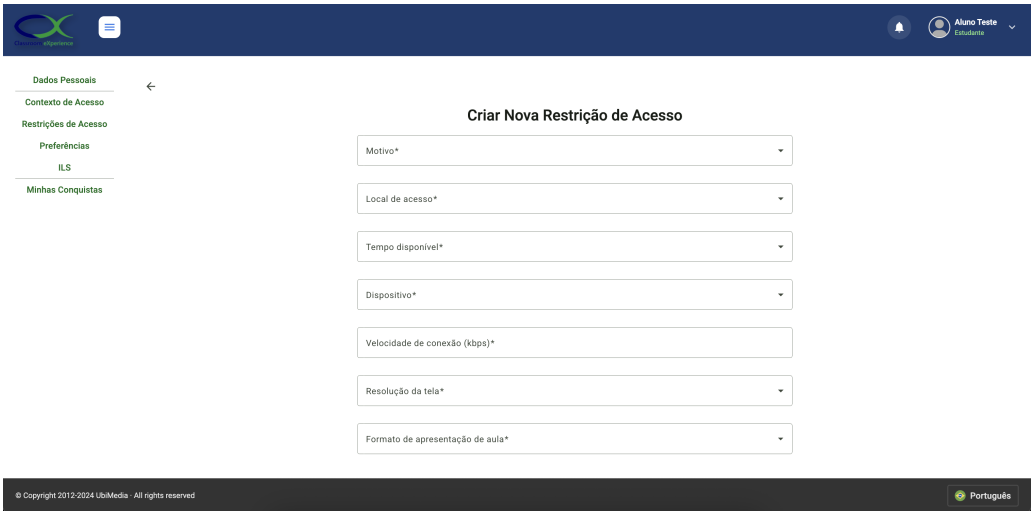


Figura 40 – Página de Criação de Restrição de Acesso [*NEW*]

Da mesma forma, as Figuras 41 e 42 ilustram a tela de Preferências. A nova disposição dos elementos oferece uma experiência mais fluida e objetiva, permitindo que o usuário configure suas preferências de maneira mais ágil.

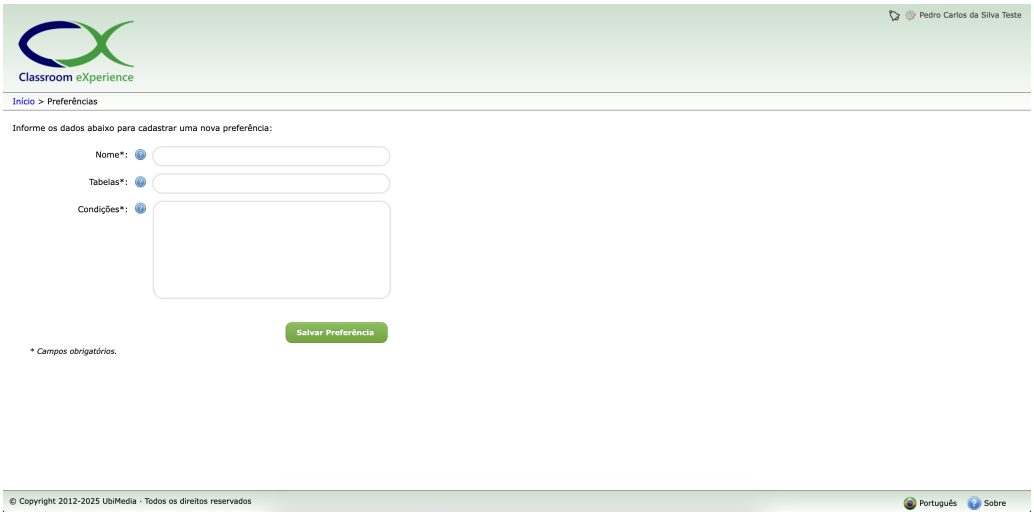


Figura 41 – Página de Preferências [*OLD*]

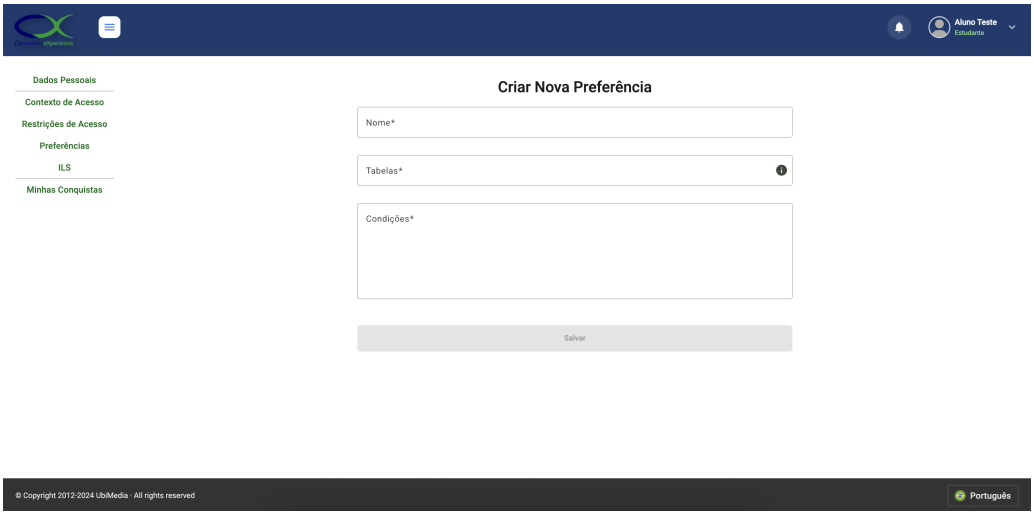


Figura 42 – Página de Preferências [*NEW*]

As Figuras 43 e 44 apresentam a tela do ILS, que foi redesenhada seguindo os mesmos princípios adotados nas demais páginas. O resultado é uma interface mais consistente, limpa e de fácil navegação.

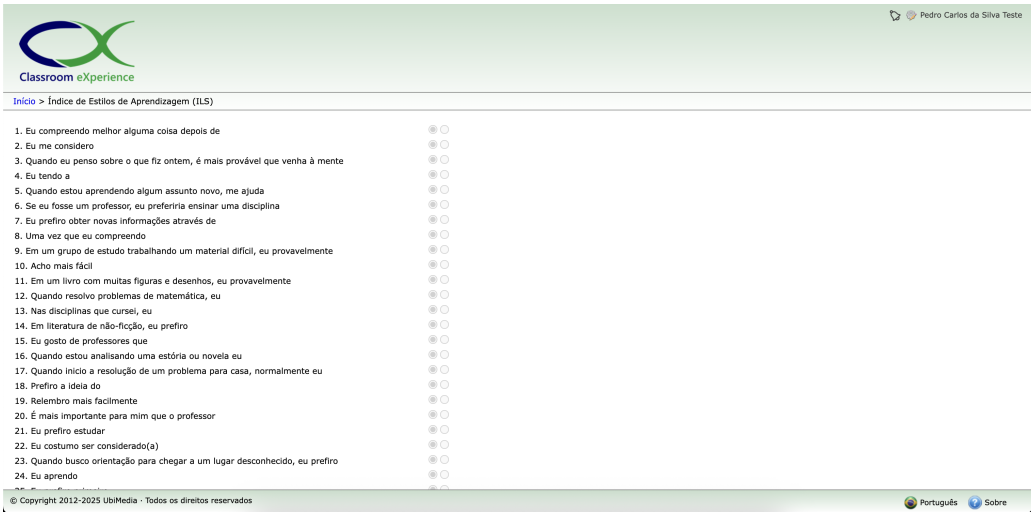


Figura 43 – Página de ILS [*OLD*]

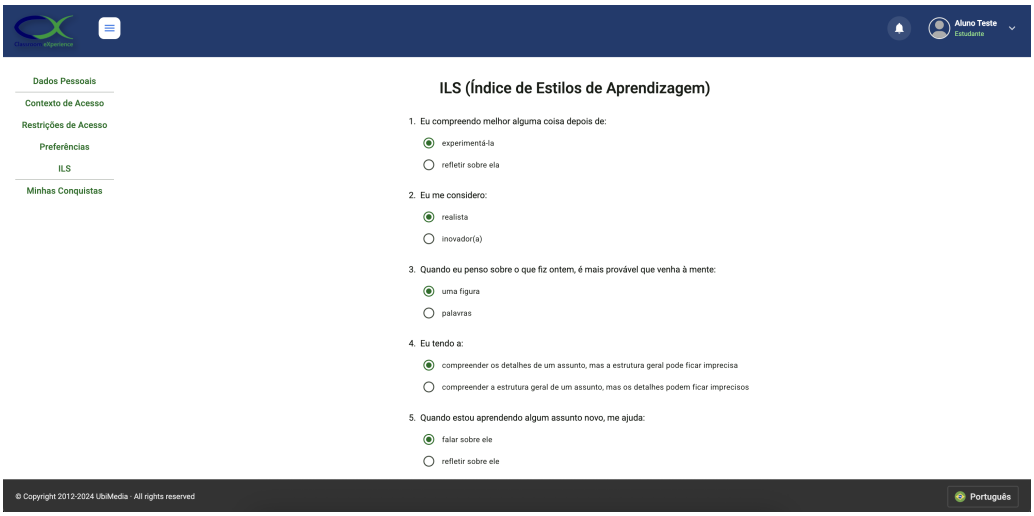


Figura 44 – Página de ILS [*NEW*]

Por fim, as Figuras 45 e 46 exibem a página de Conquistas. A nova versão mantém as funcionalidades já existentes, porém com uma apresentação mais organizada e visualmente agradável, alinhada ao novo padrão estético e funcional do sistema.

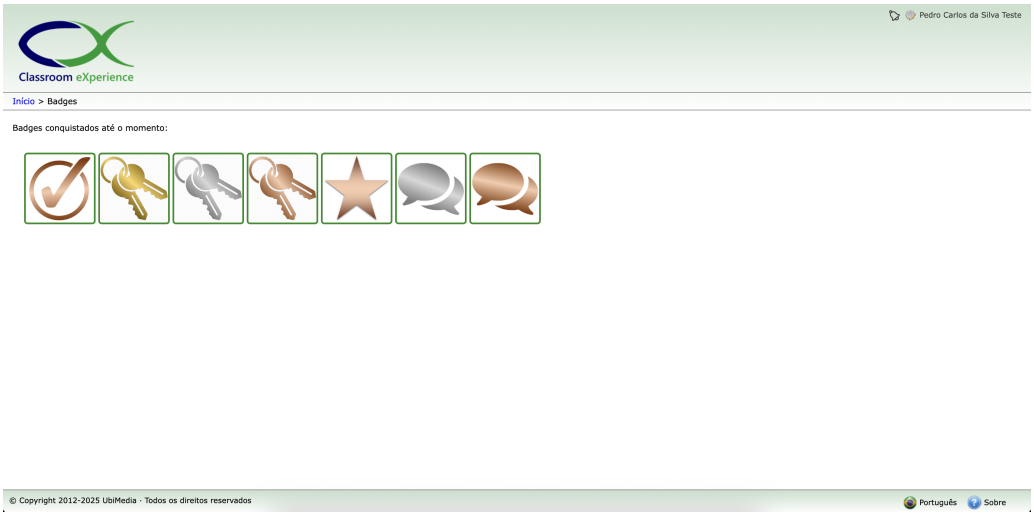


Figura 45 – Página de Conquistas [*OLD*]

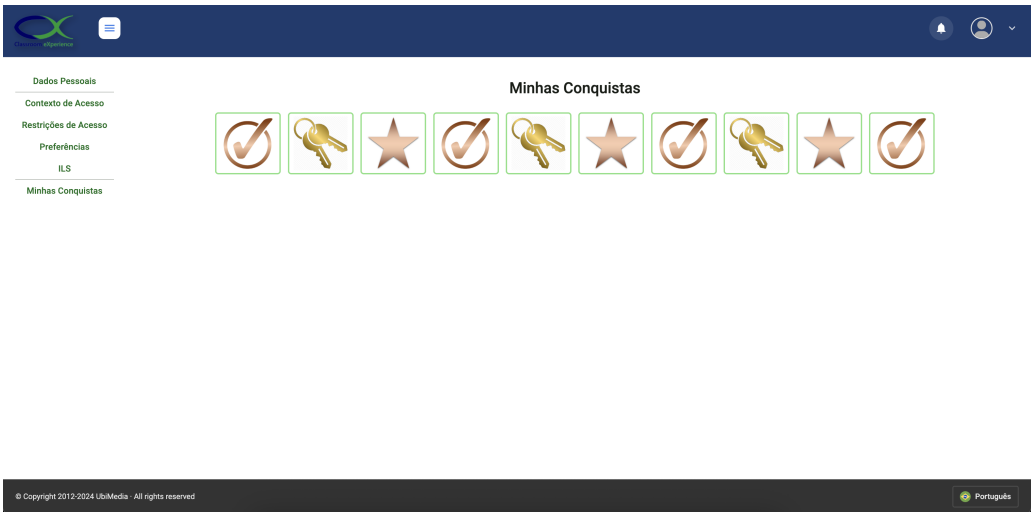


Figura 46 – Página de Conquistas [*NEW*]

Além das melhorias implementadas na versão *web*, o sistema passou a contar com uma interface responsiva para dispositivos móveis, conforme apresentado nas Figuras 47 a 52. Todas as principais funcionalidades foram cuidadosamente adaptadas, garantindo uma navegação eficiente mesmo em telas menores, com foco na usabilidade e na manutenção da consistência visual em diferentes dispositivos.

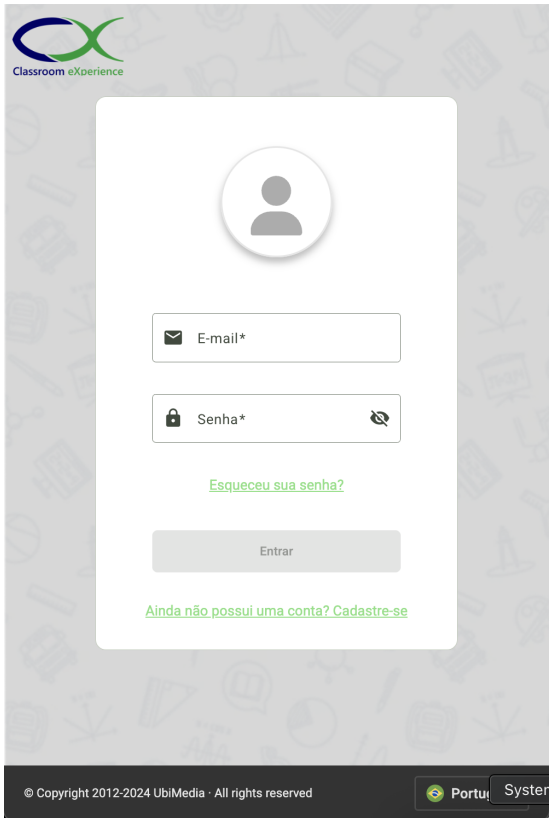


Figura 47 – Página de *Login Mobile*

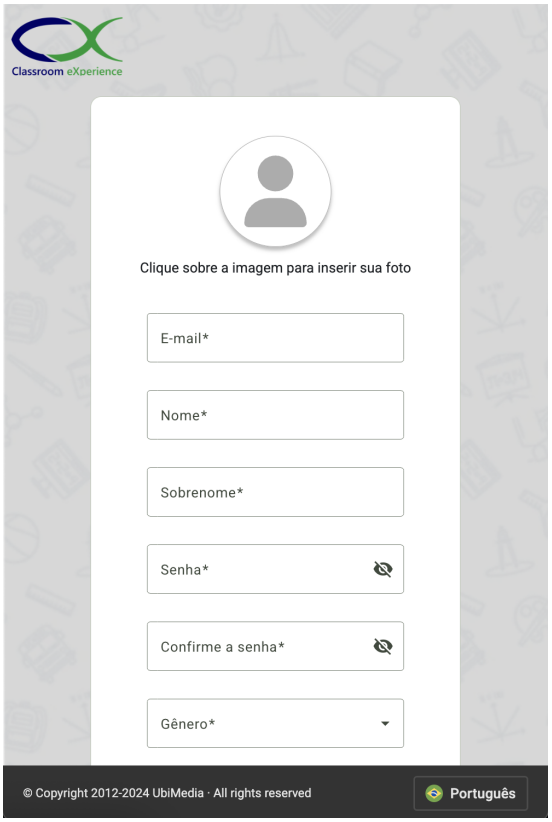


Figura 48 – Página de Registro *Mobile*

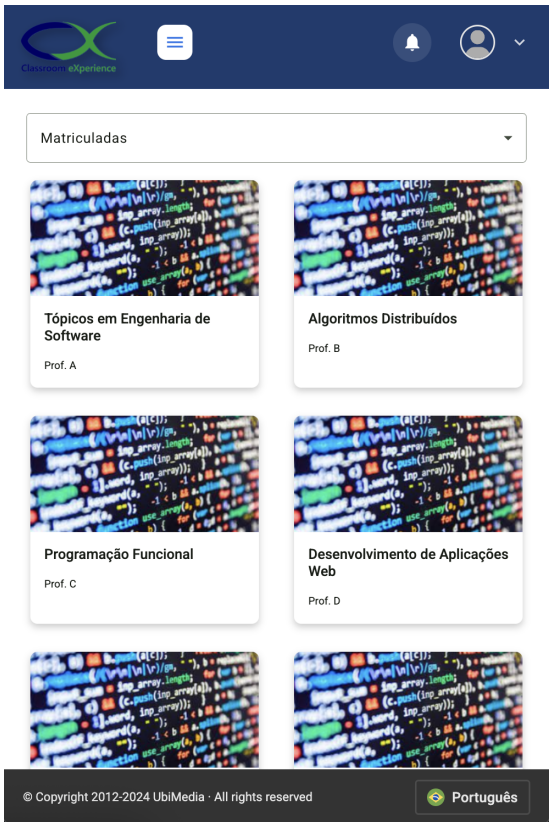


Figura 49 – Página de Disciplinas *Mobile*

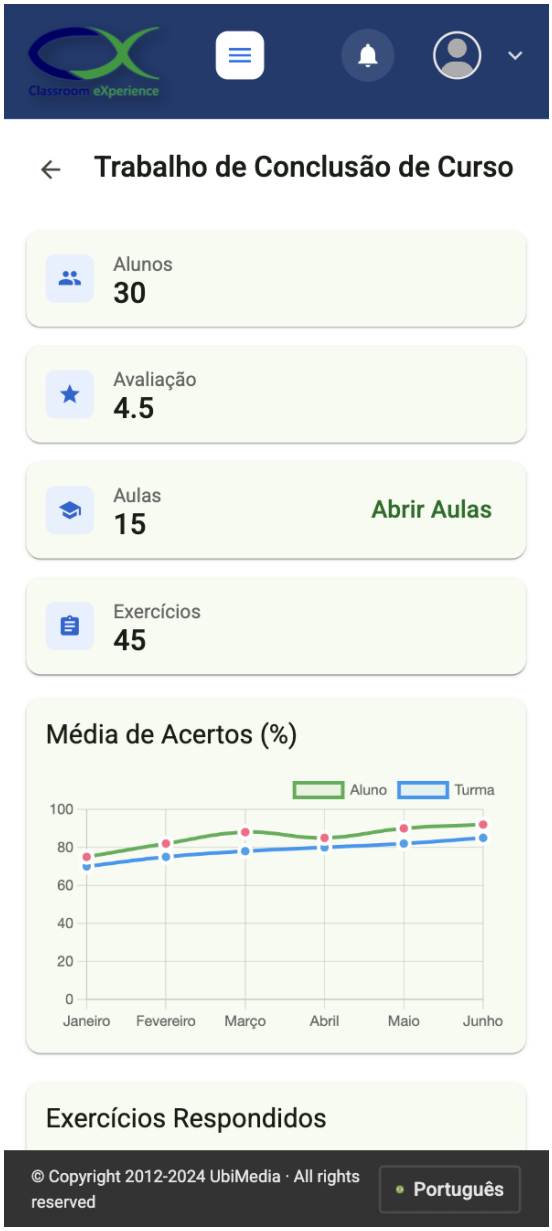


Figura 50 – Página da Disciplina *Mobile*

Figura 51 – Página de Dados Pessoais *Mobile*

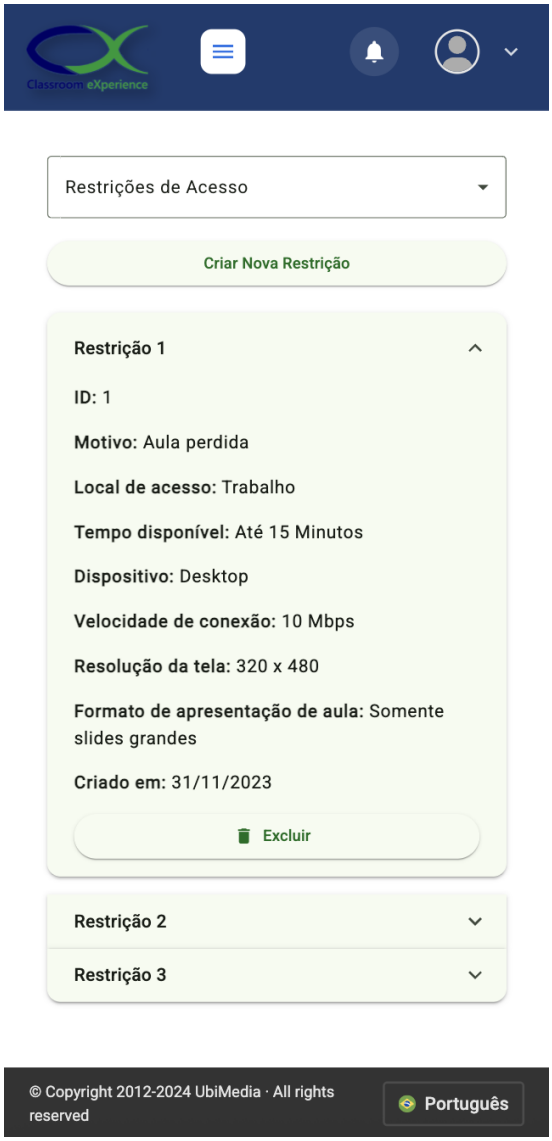


Figura 52 – Página de Restrições de Acesso *Mobile*

5.2 Resultados de Desempenho e Qualidade Técnica

Após as melhorias visuais e estruturais apresentadas na nova interface, também foi conduzida uma análise voltada à performance técnica da aplicação. Para isso, utilizou-se a ferramenta Google Lighthouse, que permite avaliar aspectos essenciais de uma aplicação *web*, como desempenho, acessibilidade, boas práticas de desenvolvimento e otimização para mecanismos de busca (SEO).

Essa análise técnica é fundamental para compreender, de forma objetiva, os ganhos proporcionados pela modernização, especialmente no que se refere à eficiência de carre-

gamento, renderização de elementos, estrutura do código e conformidade com padrões modernos da *Web*. Os testes foram realizados nas páginas de *login* do sistema, tanto em sua versão anterior quanto na versão reestruturada, contemplando os modos de exibição *mobile* e *desktop*.

A Tabela 1, em conjunto com as Figuras 53 (versão anterior) e Figura 54 (versão nova), apresenta os resultados obtidos para a versão *mobile* da página de *login*. Os dados demonstram uma melhora considerável em critérios como acessibilidade e boas práticas, com destaque para o novo sistema, que atinge pontuação máxima em conformidade com as boas práticas recomendadas.

Tabela 1 – Comparativo dos resultados do Lighthouse para a página de *login mobile*.

Versão	**Performance**	**Acessibilidade**	**Boas práticas**	**SEO**
Antiga	94	81	66	91
Nova	91	91	100	91

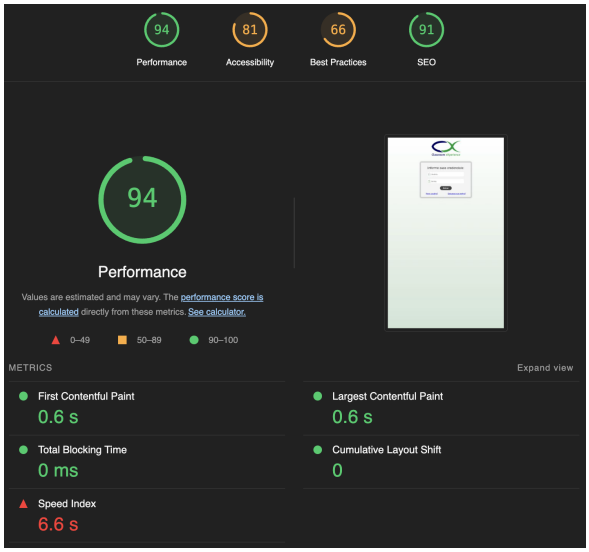


Figura 53 – Lighthouse — Página de *login* antiga (*Mobile*)

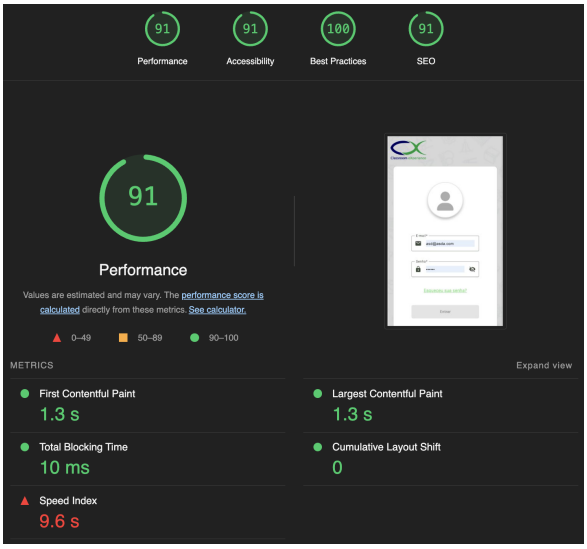


Figura 54 – Lighthouse — Página de *login* nova (*Mobile*)

A Tabela 2, em conjunto com as Figuras 55 (versão anterior) e Figura 56 (versão nova), apresenta os mesmos dados, agora para a visualização da página em modo *desktop*. Embora a performance bruta da versão antiga atinja pontuação máxima, observa-se que a versão nova apresenta resultados superiores em critérios mais amplos, como boas práticas e acessibilidade, reforçando a robustez da nova estrutura mesmo diante de pequenas variações de desempenho.

Tabela 2 – Comparativo dos resultados do Lighthouse para a página de *login desktop*.

Versão	Performance	Acessibilidade	Boas práticas	SEO
Antiga	100	81	68	91
Nova	82	91	100	91

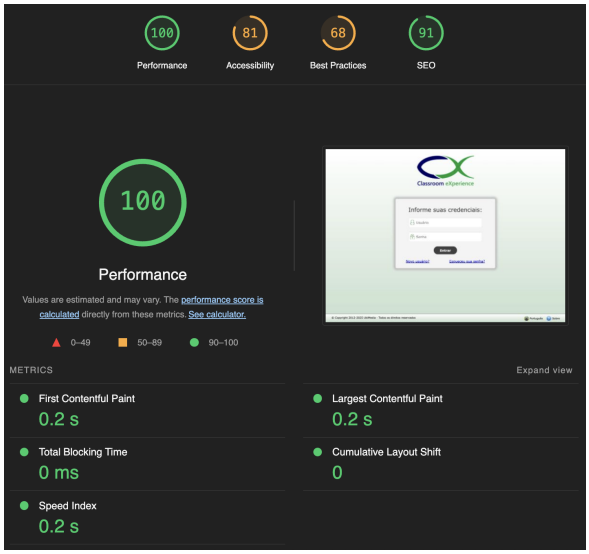


Figura 55 – Lighthouse — Página de *login* antiga (*Desktop*)

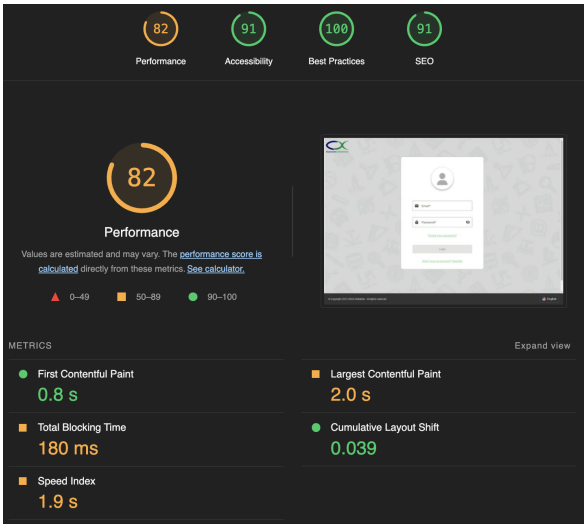


Figura 56 – Lighthouse — Página de *login* nova (*Desktop*)

Por fim, embora não tenha sido possível obter os dados de performance da página de disciplinas da versão antiga do sistema — devido à indisponibilidade técnica para sua avaliação —, apresenta-se abaixo a análise realizada na nova implementação. Essa página corresponde à tela principal da *home*, na qual o usuário visualiza todas as disciplinas disponíveis no sistema. Como mostrado nas Figuras 57 e 58, os resultados evidenciam um desempenho satisfatório tanto na versão *mobile* quanto na versão *desktop*, indicando que as práticas adotadas durante o desenvolvimento contribuíram para a construção de uma interface eficiente e alinhada às boas práticas de desenvolvimento *web*.

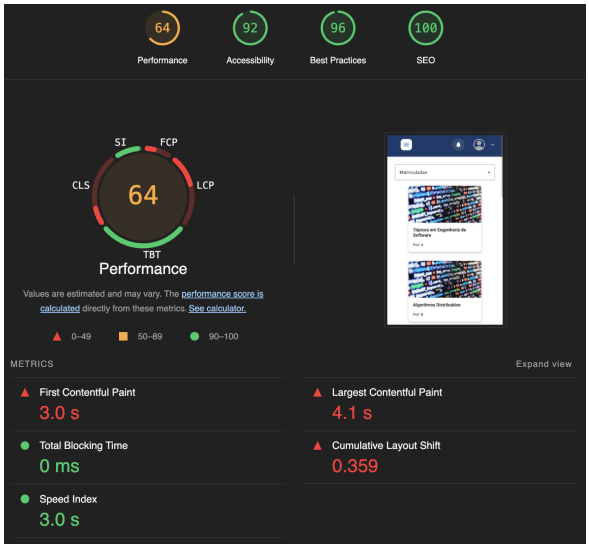


Figura 57 – Lighthouse — Página de disciplinas nova (*Mobile*)

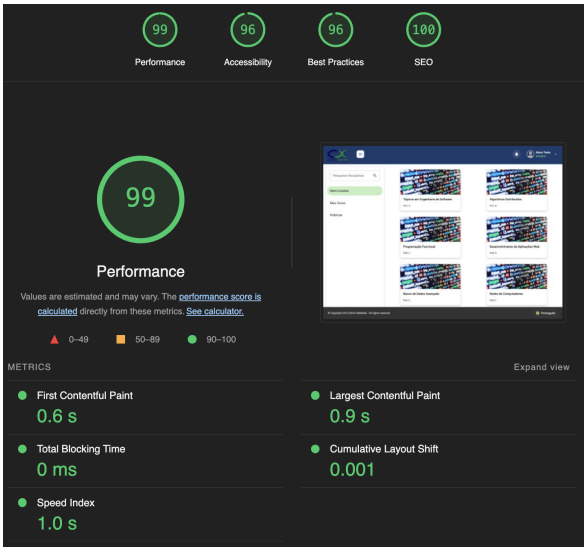


Figura 58 – Lighthouse — Página de disciplinas nova (*Desktop*)

Ainda que os resultados obtidos com a modernização tenham apresentado melhorias significativas em acessibilidade, boas práticas e SEO, observou-se uma leve queda nas métricas de performance em determinadas páginas. Esse comportamento pode ser atribuído, em grande parte, à adoção do *framework* Angular, que, apesar de suas inúmeras vantagens em organização, escalabilidade e produtividade, introduz uma camada adicional de abstração e carregamento inicial. A utilização de estratégias como *lazy loading*, injeção de dependências, inicialização de módulos e mecanismos de *change detection* pode impactar diretamente nos tempos de renderização das páginas, especialmente em ambientes com dispositivos móveis ou conexões limitadas. Tais fatores não anulam os ganhos gerais da modernização, mas indicam pontos potenciais de refinamento para futuras evoluções do sistema.

6 Conclusão

Este trabalho teve como objetivo modernizar a plataforma *Classroom eXperience (CX)*, com foco na reconstrução do *Front-End* utilizando tecnologias consolidadas no mercado e alinhadas às melhores práticas da engenharia de software. Foram abordados aspectos de arquitetura, modularidade, escalabilidade e integração contínua, além da adoção de princípios de DevOps, contribuindo não apenas para a evolução técnica da aplicação, mas também para a formação profissional envolvida no projeto.

Durante a execução, constatou-se que a migração de sistemas legados exige não apenas domínio técnico, mas uma visão abrangente do produto e de suas dependências. Algumas limitações, no entanto, foram identificadas: o sistema ainda não foi testado em ambiente de produção real, impossibilitando validações práticas com usuários finais; funcionalidades como redefinição de senha, visualização de conteúdos de aula e as voltadas ao perfil de professor permaneceram fora do escopo; e os testes automatizados se restringiram a componentes básicos, sem cobertura de fluxos mais amplos da aplicação.

A análise de desempenho também evidenciou um ponto crítico para melhorias futuras: embora as métricas de acessibilidade e boas práticas tenham melhorado significativamente, observou-se uma leve queda na performance em determinados contextos. Esse comportamento está relacionado a características intrínsecas do *framework* Angular, como o carregamento inicial de módulos e a complexidade do mecanismo de detecção de mudanças, que impactam o tempo de renderização em dispositivos menos potentes.

Apesar dessas limitações, o projeto demonstrou resultados sólidos e consistentes, com ganhos tangíveis em usabilidade, organização visual e qualidade técnica. Para as próximas etapas, recomenda-se a expansão da cobertura de testes, a revisão do código visando à otimização de performance e a inclusão das funcionalidades restantes. Testes de usabilidade com usuários reais também são fundamentais para validar a experiência proposta e promover ajustes com base em feedback concreto. Consolidando essas evoluções, a plataforma CX estará melhor preparada para atender seus usuários com mais eficiência, acessibilidade e robustez.

Referências

ARAÚJO, R.; BRANT-RIBEIRO, T.; NONATO, H.; DORçA, F.; CATTELAN, R. Segmentação colaborativa de objetos de aprendizagem utilizando bookmarks em ambientes educacionais ubíquos. In: **Anais do XXVII Simpósio Brasileiro de Informática na Educação (SBIE 2016)**. [s.n.], 2016. Disponível em: <https://www.researchgate.net/publication/309735011_Segmentacao_Colaborativa_de_Objetos_de_Aprendizagem_Utilizando_Bookmarks_em_Ambientes_Educacionais_Ubiquos>. Citado na página 13.

CATTELAN, R. G.; ARAúJO, R. D.; FERREIRA, H. N. M.; BRANT-RIBEIRO, T.; DORçA, F. A. Classroom experience: from automated multimedia capture to personalized learning. **Multimedia Tools and Applications**, 2024. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-024-20238-3>. Citado na página 12.

CLULOW, V.; BRACE-GOVAN, J. Web-based learning: experience-based research. 01 2003. Acesso em: 9 abr. 2025. Disponível em: <https://www.researchgate.net/publication/240409738_Web-based_learning_experience-based_research>. Citado na página 29.

FAETERJ-RIO. **Maior demanda de desenvolvedores Web brasileiros é para criar páginas para dispositivos móveis**. 2023. Acesso em: abr. 2025. Disponível em: <https://www.faeterj-rio.edu.br/maior-demanda-de-desenvolvedores-web-brasileiros-e-para-criar-paginas-para-dispositivos-moveis/>. Citado na página 28.

FELDER, R. M.; SILVERMAN, L. K. Learning and teaching styles in engineering education. **Engineering Education**, 1988. Disponível em: <https://www.ncsu.edu/felder-public/Papers/LS-1988.pdf>. Citado na página 13.

GeeksforGeeks. **How does the Token-Based Authentication work**. 2023. Acesso em: 02 abr. 2025. Disponível em: <https://www.geeksforgeeks.org/how-does-the-token-based-authentication-work/>. Citado na página 24.

INBEC. **A Importância da Documentação em Projetos de Software**. 2024. Acesso em: 02 abr. 2025. Disponível em: <https://inbec.com.br/blog/a-importancia-documentacao-projetos-software>. Citado na página 26.

LEON, P.; HORITA, F. Modernização de arquiteturas de sistemas para suporte à transformação digital. In: **Anais Estendidos do XVI Simpósio Brasileiro de Sistemas de Informação**. Porto Alegre, RS, Brasil: SBC, 2020. p. 61–66. Disponível em: <https://sol.sbc.org.br/index.php/sbsi_estendido/article/view/13128>. Citado na página 28.

LIM, F. J.; TAN, B. S. Overview of software re-engineering concepts, models, and approaches. **International Journal on Informatics Visualization**, Tunku Abdul Rahman University, v. 9, n. 1, p. 46–56, 2025. Disponível em: <https://www.joiv.org/index.php/joiv/article/view/3034>. Citado 2 vezes nas páginas 20 e 21.

LORD, N. **What is the Principle of Least Privilege (POLP)?** Digital Guardian, 2023. Acesso em: 02 abr. 2025. Disponível em: <https://www.digitalguardian.com/blog/what-principle-least-privilege-polp-best-practice-information-security-and-compliance>. Citado na página 24.

LUFT, M.; ROSENFELDER, S.; SYDOW, L. **State of App Marketing México & LATAM – Edição de 2021**. 2021. <https://dataai.infogram.com/pt-brazil-state-of-app-marketing-brazil-2021-1h7k230dg18pg2x>. Relatório publicado por AppsFlyer e App Annie (atualmente Data.ai). Acesso em: abr. 2025. Citado na página 28.

MARTIN, R. C. **The Clean Architecture**. The Clean Code Blog, 2012. Acesso em: 24 mar. 2025. Disponível em: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>. Citado na página 19.

Nubank Editorial. **Expandindo o desenvolvimento móvel do Nubank com o Flutter**. Nubank, 2021. Acesso em: 24 mar. 2025. Disponível em: <https://building.nubank.com.br/pt-br/expandindo-o-desenvolvimento-movel-do-nubank-com-o-flutter/>. Citado na página 17.

PICOLO, M. N. **Proposta de Modernização de Sistema de Gestão Acadêmica - SIGA**. 2021. <https://bdm.unb.br/handle/10483/30632>. Trabalho de Conclusão de Curso (Graduação em Engenharia de Software) – Universidade de Brasília, 2021. Citado 2 vezes nas páginas 22 e 28.

Rafaella Ballerini, Time Alura. **HTML, CSS e Javascript, quais as diferenças?** Alura, 2023. Acesso em: 24 mar. 2025. Disponível em: <https://www.alura.com.br/artigos/html-css-e-js-definicoes>. Citado na página 16.

RAKSI, M. **Modernizing web application: case study**. Dissertação (Dissertação (Mestrado)) — Aalto University, School of Science, Espoo, Finlândia, jul. 2017. Disponível em: <https://aaltodoc.aalto.fi/items/1f03f049-5d3b-441d-ba30-777d7bbe2b54>. Citado na página 28.

Richard Santa. **Nos próximos 5 anos o tráfego na internet será histórico**. AVI Latinoamérica, 2018. Acesso em: 30 abr. 2025. Disponível em: <https://www.avilatinoamerica.com/pt/noticia/ultimas-noticias-pt-br/357-empresas-pt-br/19618-nos-proximos-5-anos-o-trafego-na-internet-sera-historico.html>. Citado na página 12.

Robert C. Martin. **Arquitetura Limpa: o Guia do Artesão para Estrutura e Design de Software**. Rio de Janeiro: Alta Books, 2021. Citado na página 37.

SODRÉ, A. N.; MARTINS, S. T. d. S.; SIQUEIRA, M. S.; MAZZINI, S. F. Avanços tecnológicos na educação. **REVISTA DE EXTENSÃO E INICIAÇÃO CIENTÍFICA DA UNISOCIESC**, v. 8, n. 3, set. 2021. Disponível em: <https://dalfovo.com/ojs/index.php/reis/article/view/297>. Citado 2 vezes nas páginas 12 e 29.

SOUSA, N. C. d. **Redesign do software EducaTea utilizando gamificação e as diretrizes do GAIA**. Trabalho de Conclusão de Curso (Graduação) — Universidade Federal do Ceará (UFC), 2024. Disponível em: <https://repositorio.ufc.br/handle/riufc/78712>. Citado na página 29.

Tasmiha Khan. **Modernização de aplicações legadas: uma abordagem abrangente para modernizar seus negócios**. IBM, 2023. Acesso em: 25 abr. 2025. Disponível em: <https://www.ibm.com/br-pt/think/topics/legacy-application-modernization>. Citado na página 12.

web.dev Editorial. **Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage**. web.dev, 2019. Acesso em: 24 mar. 2025. Disponível em: <https://web.dev/case-studies/twitter>. Citado na página 17.

ZOLKIFLI, N. N.; NGAH, A.; DERAMAN, A. Version control system: A review. **Procedia Computer Science**, v. 135, p. 408–415, 2018. ISSN 1877-0509. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050918314819>. Citado na página 24.