

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Migrate: Software para facilitar a busca de repúblicas estudantis

Igor Laranja Borges Taquary

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Jan Mendonça Corrêa

Brasília
2025



Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Migrate: Software para facilitar a busca de repúblicas estudantis

Igor Laranja Borges Taquary

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Jan Mendonça Corrêa (Orientador)
CIC/UnB

Profa. Dra. Germana Menezes da Nobrega
CIC/UnB

Prof. Dr. Wilson Henrique Veneziano
CIC/UnB

Prof. Dr. Guilherme Novaes Ramos
Coordenador do Curso de Engenharia da Computação

Brasília, 15 de julho de 2025

Dedicatória

Dedico este trabalho à minha família, fonte de amor, apoio e inspiração. Aos meus pais, pelo sacrifício e ensinamentos; e aos meus irmãos, pela cumplicidade e motivação. Esta conquista é nossa. Com gratidão e carinho.

Agradecimentos

Agradeço a Deus pela força e sabedoria ao longo desta jornada. À minha família, que sempre me apoiou e esteve ao meu lado nessa jornada. Aos professores, pelo conhecimento e orientação que fizeram toda a diferença. E aos colegas, pela parceria, trocas de aprendizado e incentivo ao longo do caminho. Essa conquista é fruto de todos que estiveram comigo nessa caminhada.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Este projeto teve como objetivo desenvolver um software para um problema real enfrentado por estudantes que precisam mudar de cidade para ingressar no ensino superior: a dificuldade de encontrar moradias adequadas, seguras e acessíveis. A aplicação *web* proposta visa simplificar a busca por repúblicas estudantis, oferecendo funcionalidades como filtros por proximidade das instituições de ensino, opções de moradias compartilhadas por gênero e descrições detalhadas das acomodações. A motivação para o projeto surgiu da observação de casos reais de estudantes que enfrentaram desafios ao se estabelecer em novas cidades, muitas vezes resultando em desistências no primeiro ano do curso.

Para o desenvolvimento da solução, foram adotadas metodologias como *Test-Driven Development (TDD)*, arquitetura cliente-servidor, *Lean e Mobile First*, para obter um processo ágil, eficiente e focado nas necessidades do usuário. A aplicação foi implementada utilizando tecnologias modernas, como o *framework NestJS* para o *back-end*, devido à sua escalabilidade e modularidade, e o *ReactJS* para o *front-end*, assegurando uma interface intuitiva e responsiva, com reutilização de código. Além disso, foram realizados testes unitários com o *Jest* para garantir a qualidade e a confiabilidade do sistema.

Este texto relata o processo de desenvolvimento de uma solução completa, desde a percepção do problema até a publicação do software, permitindo a compreensão das diversas etapas, metodologias e ferramentas necessárias para alcançar o objetivo final. Como resultados, espera-se que a plataforma Migrate melhore a experiência de calouros na busca por acomodações, reduza a evasão acadêmica e promova uma transição mais tranquila e segura para a vida universitária. Futuramente, o sistema pode ser expandido com funcionalidades adicionais, como integração com serviços de transporte e suporte à comunidade universitária, ampliando seu impacto positivo.

Palavras-chave: moradia estudantil, plataforma web, redução de evasão acadêmica, desenvolvimento de software, design centrado no usuário

Abstract

This project aimed to develop software to address a real problem faced by students who need to move to a new city to attend higher education: the difficulty of finding suitable, safe, and affordable housing. The proposed web application simplifies the search for student housing by offering features such as filters for proximity to educational institutions, gender-specific shared housing options, and detailed accommodation descriptions. The motivation for the project arose from observing real cases of students who faced challenges when settling in new cities, often leading to dropouts during the first year of their courses.

To develop the solution, methodologies such as Test-Driven Development (TDD), client-server architecture, Lean and Mobile First approaches were adopted to ensure an agile, efficient, and user-focused process. The application was implemented using modern technologies, including the NestJS framework for the back-end, due to its scalability and modularity, and ReactJS for the front-end, ensuring an intuitive and responsive interface with code reusability. Additionally, unit testing was conducted using Jest to guarantee system quality and reliability.

This text documents the development process of a complete solution, from problem identification to software deployment, providing insight into the various stages, methodologies, and tools required to achieve the final goal. As a result, the Migrate platform is expected to improve the housing search experience for incoming students, reduce academic dropout rates, and promote a smoother and safer transition to university life. In the future, the system can be expanded with additional features, such as integration with transportation services and support for university communities, further enhancing its positive impact.

Keywords: student housing, web platform, academic dropout reduction, software development, user-centered design

Sumário

1 Introdução **1**

1.1 Problema . . . 2
1.2 Motivação . . . 3
1.3 Objetivo . . . 3
1.3.1 Objetivos Específicos . . . 3
1.3.2 Estrutura dos Capítulos . . . 4

2 Fundamentação teórica **5**

2.1 Migração Estudantil . . . 5
2.2 Ferramentas Semelhantes . . . 6
2.3 Metodologias Utilizadas . . . 8
2.3.1 Lean . . . 8
2.3.2 Test-Driven Development . . . 8
2.3.3 Arquitetura Cliente-Servidor . . . 9
2.3.4 Mobile First . . . 10
2.4 Ferramentas e Tecnologias . . . 11
2.4.1 TypeScript . . . 11
2.4.2 ReactJS . . . 11
2.4.3 Bootstrap . . . 13
2.4.4 NestJS . . . 13
2.4.5 Swagger . . . 14
2.4.6 Jest . . . 14
2.4.7 MinIO . . . 14
2.4.8 MySQL . . . 15
2.4.9 Google Maps . . . 15
2.4.10 Docker . . . 15
2.4.11 Figma . . . 15
2.5 Criptografia e Segurança . . . 16
2.5.1 JSON Web Token . . . 16

2.5.2 Hash de Senhas com Salt . . . 17
2.5.3 Considerações sobre Segurança . . . 17
2.6 Lei Geral de Proteção de Dados (LGPD) . . . 18
2.6.1 Impactos da LGPD no Desenvolvimento de Aplicações Web . . . 18
2.6.2 Medidas de Segurança e Boas Práticas . . . 19
2.6.3 Conformidade e Riscos Jurídicos . . . 19

3 Desenvolvimento **21**
3.1 Análise de Requisitos . . . 21
3.1.1 Introdução . . . 21
3.1.2 Requisitos Funcionais . . . 22
3.1.3 Requisitos Não Funcionais . . . 23
3.2 Prototipação de telas . . . 24
3.2.1 Fluxograma . . . 25
3.2.2 Protótipos . . . 26
3.3 Configurações Iniciais . . . 31
3.4 Banco de Dados . . . 32
3.4.1 Modelagem . . . 32
3.4.2 Normalização do Banco de Dados . . . 34
3.4.3 Diagrama Entidade-Relacionamento (ER) . . . 35
3.4.4 Implementação . . . 35
3.5 Implementação Servidor . . . 39
3.5.1 Módulos . . . 39
3.5.2 Testes Unitários . . . 41
3.6 Implementação Cliente . . . 45
3.6.1 Estrutura do Código . . . 45
3.6.2 Contextos . . . 46
3.6.3 Roteamento de Páginas . . . 46
3.6.4 Responsividade e Compatibilidade . . . 49

4 A Aplicação Migrate **51**
4.1 Autenticação . . . 52
4.2 Cadastro de Anúncios . . . 55
4.3 Busca de Anúncios . . . 59
4.4 Visualização de Anúncios . . . 60
4.5 Edição de Anúncio . . . 62
4.6 Demonstração Prática . . . 62

5 Conclusões **64**
5.1 Trabalhos Futuros 65

Referências **66**

Apêndice **67**

A Política de Privacidade **68**

Lista de Figuras

3.1 Fluxograma de navegação - LucidChar. 25
3.2 Protótipo da tela inicial. 26
3.3 Protótipo da tela de cadastro de usuário. 27
3.4 Protótipo da tela de login. 27
3.5 Protótipo de tela de cadastro de acomodação. 28
3.6 Protótipo de tela de acomodação. 30
3.7 Protótipo de tela de busca de acomodação. 30
3.8 Diagrama de entidade-relacionamento dos dados - MySQL Workbench. . . 36
3.9 Exemplo de código de criação de entidade com TypeORM. 38
3.10 Código do módulo principal *NestJS* com devidas injeções de dependências. 40
3.11 Código de configuração do módulo NestJS de JSON Web Token. 41
3.12 Código do método de login do módulo de autenticação. 42
3.13 Código do middleware de validação de JSON Web Token. 43
3.14 Código de teste unitário do método de busca de acomodações. 44
3.15 Relatório gerado pelo Jest da cobertura de teste do projeto. 44
3.16 Código de gerenciamento de rotas no cliente usando react-router-dom. . . . 47
3.17 Tela inicial da aplicação em dispositivo com tela pequena. 50

4.1 Formulário de cadastro apontando erros de preenchimento. 52
4.2 Formulário de cadastro com dados corretamente preenchidos. 53
4.3 Aviso de conta criada com sucesso. 53
4.4 Formulário de autenticação preenchido. 54
4.5 Tela de perfil do usuário logado. 54
4.6 Swagger UI da aplicação exibindo rotas de autenticação e de usuário. . . . 54
4.7 Tela de anúncios de acomodações do usuário. 55
4.8 Primeira etapa do formulário de cadastro de anúncio de acomodação. . . . 55
4.9 Formulário de cadastro de acomodação preenchido. 56
4.10 Etapa de cadastro da localização da acomodação. 57
4.11 Etapa de cadastro de fotos com erro. 58
4.12 Etapa de cadastro de fotos preenchido com 3 imagens. 58

4.13 Página de explorar acomodações cadastradas. 59
4.14 Página de detalhes da acomodação. 60
4.15 Mapa com rotas da acomodação à instituição. 61
4.16 Tela “Meus Anúncios” com um anúncio cadastrado. 62

Lista de Tabelas

2.1 Comparativo entre o desenvolvimento tradicional e o desenvolvimento utilizando TDD. 9

3.1 Tipos de tarefas do quadro Kanban 31

Lista de Abreviaturas e Siglas

CEP Código de Endereçamento Postal.

DTO Data Transfer Object.

JWT JSON Web Token.

SISU Sistema de Seleção Unificada.

TDD Test-Driven Development.

URL Uniform Resource Locator.

UUID Universally unique identifier.

Capítulo 1

Introdução

O aumento significativo no número de vagas em instituições de ensino superior, promovido pelo Sistema de Seleção Unificada (SISU), gerou um fluxo de estudantes para diferentes cidades [1]. No entanto, a falta de moradias estudantis adequadas dificulta a adaptação de estudantes que migram para novas cidades, especialmente aqueles com recursos financeiros limitados.

Aluguéis elevados e a ausência de sistemas eficientes para encontrar acomodações seguras e acessíveis agravam a vulnerabilidade econômica, comprometendo a qualidade de vida acadêmica desses jovens, destacando a necessidade de soluções que facilitem esse processo. Este trabalho, portanto, propõe o desenvolvimento de uma aplicação *web* para auxiliar calouros a encontrarem alojamentos, facilitando sua transição para novas cidades e garantindo uma experiência universitária mais tranquila. Diferente de plataformas genéricas de aluguel, o aplicativo proposto é especializado no público universitário, oferecendo funcionalidades como filtros por proximidade das instituições de ensino, opções de moradias que permitem a divisão por gênero e descrições detalhadas das acomodações, adaptadas às necessidades dos estudantes. Essas funcionalidades visam simplificar a busca por moradias que atendam a critérios essenciais, como localização estratégica, segurança e compatibilidade com o perfil do usuário, garantindo uma experiência mais assertiva e personalizada.

A plataforma busca preencher uma lacuna existente no contexto educacional brasileiro, em que a migração estudantil é frequentemente marcada pela dificuldade de acesso a informações sobre moradias seguras, acessíveis e próximas às instituições de ensino. Ela é voltada especialmente para estudantes de baixa renda, oriundos de diferentes regiões do país, que enfrentam desafios adicionais ao se deslocarem para cidades onde o custo de vida é elevado e a oferta de moradias estudantis é insuficiente. Estudos demonstram que a ausência de suporte adequado durante esse processo pode contribuir para a evasão acadêmica e afetar o desempenho dos estudantes [2].

Logo, este projeto objetiva analisar o problema e as soluções já disponíveis no mercado, evidenciando suas limitações. Além disso, busca-se demonstrar como a implementação de uma plataforma centralizada pode não apenas facilitar a busca por moradias, mas também contribuir para a redução da evasão acadêmica e a promoção de uma experiência universitária mais inclusiva e equitativa. Serão detalhadas as tecnologias e metodologias utilizadas no desenvolvimento do sistema, em seguida descritas a análise de requisitos, prototipação de telas, modelagem de dados e a codificação do software.

Ao longo do trabalho, serão abordados os desafios enfrentados no planejamento e implementação do software, desde a prototipação e modelagem do banco de dados até a realização de testes automatizados para assegurar a qualidade do sistema. Para o desenvolvimento da plataforma, serão utilizadas tecnologias modernas e amplamente adotadas no mercado, como o *framework NestJS* para o *back-end*, devido à sua escalabilidade e modularidade, e *ReactJS* para o *front-end*, garantindo uma interface intuitiva e responsiva. A escolha dessas ferramentas visa garantir um sistema robusto, de fácil manutenção e adaptável às necessidades dos usuários. Por fim, são apresentados os resultados obtidos e sugestões para trabalhos futuros, com o objetivo de aprimorar a experiência dos estudantes e contribuir para a redução das dificuldades associadas à migração estudantil no Brasil.

1.1 Problema

O cenário enfrentado por muitos estudantes que mudam para uma nova cidade em busca de acessar o ensino superior, muitas vezes distante da residência de origem, representa um desafio significativo no contexto educacional. A falta de familiaridade com a nova localidade, somada à ausência de redes de apoio e à insegurança em relação às opções de moradia, pode tornar essa transição uma experiência estressante. Para muitos calouros, estabelecer contato com veteranos ou residentes locais que possam oferecer orientações sobre acomodações adequadas é uma tarefa complicada, e somado a outros problemas relacionados à migração, muitos estudantes acabam desistindo do curso durante o primeiro ano de graduação [3]. Portanto, a inexistência de um sistema centralizado e eficiente para a busca de moradia reforça a necessidade de soluções práticas e acessíveis para contornar esse problema.

Adicionalmente, pesquisas [3] que analisaram o impacto do SISU sobre a mobilidade estudantil revelam aspectos relevantes desse cenário. A implementação do sistema contribuiu significativamente para o aumento da migração de estudantes entre municípios e estados, ampliando o acesso à educação superior. No entanto, os dados também evidenciam um crescimento expressivo nas taxas de evasão acadêmica, indicando que os

desafios enfrentados por estudantes em transição, como a busca por moradias adequadas e a adaptação a novos ambientes, ainda demandam atenção e soluções específicas.[3]

Além disso, a carência de suporte adequado na busca por alojamento pode levar os estudantes a tomar decisões precipitadas. Moradias que, à primeira vista, parecem convenientes podem não atender a critérios essenciais de segurança, acessibilidade e conforto. A hipótese é que a escolha inadequada de moradia pode resultar não apenas em gastos excessivos, mas também em situações de vulnerabilidade e estresse, comprometendo a experiência acadêmica e pessoal. Logo, o desenvolvimento de soluções tecnológicas que auxiliem os estudantes nesse processo é um ponto importante.

1.2 Motivação

A ideia inicial para este projeto surgiu da observação de casos reais de pessoas próximas, como meu irmão, primos e amigos que, ao ingressarem em uma universidade em outra cidade, enfrentaram dificuldades para encontrar uma moradia segura e acessível. Relatos semelhantes de outros conhecidos reforçaram a percepção de que essa é uma problemática comum entre estudantes universitários.

O crescente movimento de estudantes para diferentes cidades em busca de uma educação superior acessível evidencia a necessidade de soluções inovadoras que garantam uma transição segura e eficiente. A ausência de plataformas integradas que facilitem o processo de busca por alojamentos acessíveis e adequados compromete o bem-estar dos estudantes e pode gerar impactos negativos em sua adaptação. Nesse contexto, surge a oportunidade de aliar tecnologia e acessibilidade para criar uma ferramenta que centralize informações, simplifique processos e promova a inclusão de estudantes em novas localidades, fortalecendo sua trajetória acadêmica desde o início.

1.3 Objetivo

Desenvolver um aplicativo *web* especializado na busca de alojamentos para estudantes universitários, com foco em oferecer uma solução centralizada e eficiente que facilite a adaptação de calouros em novas cidades, promovendo segurança, conforto e acessibilidade.

1.3.1 Objetivos Específicos

Em busca de alcançar o objetivo central do trabalho, foram necessários os seguintes objetivos específicos:

- Identificar e compreender os principais desafios enfrentados por estudantes universitários, especialmente calouros, ao se mudarem para novas localidades em busca de alojamento, incluindo aspectos como acessibilidade, mobilidade, custo e rede de apoio;
- Implementar uma aplicação *web* completa com tecnologias modernas e amplamente utilizadas;
- Desenvolver uma interface intuitiva e de fácil uso, que contemple diferentes níveis de familiaridade tecnológica entre os estudantes, promovendo acessibilidade e inclusão;
- Desenvolver uma aplicação segura, reduzindo riscos de ações não autorizadas e vazamentos.

1.3.2 Estrutura dos Capítulos

Os capítulos do trabalho são estruturados da seguinte forma:

- **Capítulo 1: Introdução**
  O capítulo apresenta uma visão geral do projeto, incluindo os objetivos, a justificativa e a organização do documento.
- **Capítulo 2: Fundamentação teórica**
  O capítulo apresenta os principais conceitos, metodologias e ferramentas que embasam o projeto, fornecendo o contexto necessário para a sua compreensão.
- **Capítulo 3: Desenvolvimento**
  Este capítulo detalha a implementação do projeto, abordando desde a modelagem do banco de dados até o desenvolvimento do software e os testes realizados.
- **Capítulo 4: Conclusão**
  Este capítulo consolida as etapas do projeto, discute as implicações dos resultados gerados pela aplicação e sugere possíveis direções para trabalhos futuros.

Capítulo 2

Fundamentação teórica

Este capítulo tem como objetivo apresentar conceitos relevantes para o desenvolvimento do trabalho, analisar e comparar aplicações existentes voltadas ao compartilhamento de moradia, e também descrever as ferramentas e tecnologias utilizadas no desenvolvimento da aplicação.

2.1 Migração Estudantil

Estudos internacionais demonstram que a distância entre a residência do estudante e a instituição de ensino é um fator determinante na decisão de migração. A literatura aponta que a decisão de mudar-se para outra localidade para cursar o ensino superior envolve uma análise complexa dos custos e benefícios, onde a distância desempenha um papel crucial [1]. A migração estudantil, portanto, não é apenas uma questão de deslocamento físico, mas também envolve custos financeiros e psicológicos significativos. O estudante precisa considerar despesas com moradia, deslocamentos e visitas à família, além do impacto emocional de deixar sua rede de apoio.

No Brasil, a situação é particularmente desafiadora devido à falta de mecanismos eficientes para auxiliar os estudantes na busca por moradia adequada. A ausência de um sistema centralizado que facilite essa busca pode levar a decisões precipitadas sobre alojamento, resultando em situações que não atendem aos critérios essenciais de segurança, acessibilidade e conforto. Muitos estudantes enfrentam dificuldades para encontrar moradias que atendam às suas necessidades, o que pode afetar negativamente sua adaptação e desempenho acadêmico [1].

Além das questões práticas, a migração estudantil também é influenciada por fatores sociais e emocionais. A necessidade de deixar uma rede de apoio familiar e social pode aumentar o custo psicológico da migração, tornando a transição para uma nova cidade ainda mais desafiadora. Estudos mostram que estudantes de estratos sociais menos favo-

recidos são mais propensos a se situar em áreas distantes das universidades, o que pode limitar seu acesso a recursos e apoio [1]. Essa dinâmica revela a necessidade urgente de soluções que não apenas ajudem na busca por moradia, mas que também ofereçam suporte emocional e social aos estudantes em processo de adaptação.

Diante dos desafios evidentes na busca por moradia enfrentados pelos estudantes que se deslocam para novas localidades, é crucial entender como essas dificuldades são abordadas pelas ferramentas existentes. Para aprimorar a solução proposta, que é o desenvolvimento de um aplicativo *Web* especializado na busca de alojamentos para estudantes universitários, é necessário realizar uma pesquisa de mercado detalhada.

Essa pesquisa visa explorar as soluções já disponíveis e avaliar suas funcionalidades e limitações. O objetivo é identificar as principais falhas e lacunas das aplicações atuais que atendem à busca por moradia estudantil. A análise das ferramentas existentes permitirá compreender as formas como os estudantes utilizam essas plataformas, os aspectos que consideram mais relevantes e as áreas onde as soluções atuais falham em atender às suas necessidades.

Ao investigar essas aplicações, será possível identificar quais características e funcionalidades são mais valorizadas pelos usuários e quais aspectos ainda precisam ser aprimorados. Com base nessa pesquisa, o novo aplicativo poderá ser desenvolvido para preencher essas lacunas, oferecendo uma solução mais eficaz e adaptada às reais necessidades dos estudantes que se mudam para novas cidades. Assim, a pesquisa de mercado não apenas justifica a necessidade de um novo aplicativo, mas também orienta o desenvolvimento de uma ferramenta que possa proporcionar uma experiência mais eficiente e satisfatória na busca por moradia estudantil.

2.2 Ferramentas Semelhantes

Para compreender melhor as estratégias utilizadas pelos estudantes na busca por moradia em outro estado, foi realizada uma análise de ferramentas existentes que possuem essa finalidade. Através de pesquisas em buscadores na internet (*Google, Bing, DuckDuckGo*) por termos como "repúblicas", "quarto para alugar", "moradia estudantil", foram selecionadas algumas dessas ferramentas e identificados os principais pontos de falha relacionados à busca por moradias estudantis. Essa avaliação permite um entendimento mais claro dos aspectos positivos e negativos das soluções atualmente disponíveis no mercado, servindo como base para o direcionamento do projeto proposto. Nesse contexto foram analisadas as seguintes ferramentas e destacados os pontos de falha:

1. **Salter**: Apresenta uma navegação pouco intuitiva, além de conter anúncios desatualizados e com informações limitadas sobre repúblicas. Embora permita o cadastro

de quartos para locação, o processo é prejudicado pela ausência de fotos e pela disponibilização insuficiente de detalhes sobre a localização.

2. **Repúblicas Federais**: A plataforma funciona como um chat onde os usuários podem publicar comentários em busca de repúblicas ou para divulgar vagas. No entanto, a navegação também é pouco intuitiva, e os comentários são exibidos em ordem cronológica decrescente, o que pode dificultar a visualização de informações relevantes. Outro ponto crítico é a exposição pública dos dados de contato, que ficam acessíveis a qualquer usuário, mesmo que não autenticado.

3. **Airbnb**: Embora seja uma plataforma consolidada, é voltada principalmente para hospedagens temporárias, com cobrança por diária e foco em viagens de lazer ou trabalho. Alguns estudantes utilizam o Airbnb ao chegar em uma nova cidade, enquanto procuram por uma moradia fixa. Contudo, não atende às necessidades específicas de estudantes que buscam soluções de longo prazo.

4. **WebQuarto**: Trata-se de uma plataforma direcionada ao aluguel de quartos e apartamentos para serem compartilhados. No entanto, não oferece informações sobre as instituições de ensino próximas ou as opções de transporte público, o que pode dificultar a escolha de acomodações adequadas e bem localizadas, em relação a sua instituição de ensino, para estudantes que acabaram de chegar à cidade.

5. **Grupos em Redes Sociais**: Uma prática bastante comum entre estudantes é a utilização de grupos em redes sociais, como Facebook, WhatsApp e Telegram, para encontrar quartos e moradias. Essa abordagem oferece um contato mais dinâmico entre os usuários e uma filtragem básica dos participantes. Contudo, há desvantagens significativas, como a falta de organização das informações e a necessidade de os usuários estarem constantemente ativos para obterem sucesso em suas buscas. Por exemplo, um dos grupos destinados para este fim no Facebook, Republicas da UnB, conta com quase 11 mil membros, onde os anúncios de moradias se misturam com postagens de doações, dúvidas e solicitações para outros grupos.

Tendo em vista as ferramentas analisadas, seus defeitos e qualidades, é possível projetar um software especializado para a busca de repúblicas estudantis, com uma boa usabilidade, que atenda às necessidades específicas dos estudantes, como a localização próxima às instituições de ensino e opções de transporte público, além de oferecer uma navegação intuitiva e informações detalhadas sobre as moradias disponíveis. A plataforma proposta deve evitar os pontos críticos identificados nas ferramentas existentes, como a exposição pública de dados de contato e a falta de organização das informações, proporcionando uma experiência mais segura e eficiente para os usuários. Além disso, a nova solução deve

ser capaz de filtrar e organizar os anúncios de forma clara e acessível, evitando a dispersão de conteúdo e facilitando a busca por moradias adequadas. Com base nessa análise, o desenvolvimento de uma aplicação especializada pode preencher as lacunas identificadas, oferecendo uma alternativa mais robusta e direcionada ao público estudantil.

2.3 Metodologias Utilizadas

Nesta seção, detalham-se as metodologias empregadas no desenvolvimento do software, considerando tanto o processo de gerenciamento quanto as práticas de codificação.

2.3.1 Lean

Para o gerenciamento do projeto, foi adotada a metodologia *Lean* [4], promovendo a eficiência e otimização do fluxo de trabalho. O *Lean* é uma abordagem focada na identificação e eliminação de desperdícios, como tarefas desnecessárias, retrabalho ou espera, visando maximizar o valor gerado pelo processo de desenvolvimento. Além disso, a metodologia enfatiza a melhoria contínua e a entrega consistente de resultados alinhados aos objetivos do projeto.

Durante o desenvolvimento, os princípios *Lean* foram aplicados para mapear o fluxo de valor, ou seja, todas as etapas necessárias para transformar os requisitos do software em funcionalidades prontas. Isso permitiu identificar gargalos e atividades que não contribuíam diretamente para o progresso do projeto. Para garantir um fluxo de trabalho equilibrado, o sistema "puxado"foi utilizado, em que novas tarefas só eram iniciadas quando havia capacidade disponível, evitando sobrecargas e garantindo um ritmo sustentável.

A visualização do fluxo de trabalho foi facilitada por meio de ferramentas como o Trello, que permitiu organizar as tarefas em colunas representando etapas do processo, como *A Fazer*, *Em Progresso* e *Concluído*. Essa abordagem possibilitou uma gestão mais eficiente do tempo e dos recursos, além de priorizar as funcionalidades mais relevantes para o andamento do projeto. Adotando uma postura de melhoria contínua, revisando periodicamente o processo e propondo ajustes para aumentar a produtividade e a qualidade do software desenvolvido.

2.3.2 Test-Driven Development

A prática de *Test-Driven Development (TDD)* [5] foi utilizada como abordagem principal para a construção do código do *back-end*. O TDD envolve a criação de testes automatizados antes mesmo do desenvolvimento da funcionalidade em si, garantindo que o código produzido seja confiável e esteja alinhado com os requisitos.

Esta metodologia incentiva a escrita de código mais limpo e bem estruturado, pois os testes ajudam a detectar erros e encorajam refatorações contínuas. Como os testes são escritos antes do código, eles garantem que o desenvolvedor pense nos possíveis erros e casos extremos desde o início, evitando *bugs* mais adiante no processo.

Além disso, os testes servem como uma espécie de documentação que descreve como o código deve se comportar. Isso facilita a compreensão de como as funcionalidades foram implementadas. Permitindo que os desenvolvedores refatorem o código com mais segurança.

O processo de TDD seguido neste projeto envolveu os seguintes passos:

1. **Escrever um teste automatizado que falhe inicialmente**, pois a funcionalidade ainda não foi implementada.
2. **Desenvolver o código necessário para que o teste passe**, garantindo que a funcionalidade seja entregue de acordo com os requisitos.
3. **Refatorar o código**, melhorando sua estrutura e qualidade, sem alterar seu comportamento, para assegurar a manutenibilidade e escalabilidade do software.

A utilização de ferramentas como o *Jest* 2.4.6, uma biblioteca de testes para *JavaScript*, é fundamental para a aplicação do TDD. Permitindo a criação de testes unitários e de integração que asseguraram a qualidade e a estabilidade do sistema.

Comparado ao desenvolvimento tradicional utilizar o TDD é mais desafiador inicialmente, todavia recompensador a longo prazo. Sendo um grande diferencial para aplicações mais complexas, com entregas contínuas e equipes com maior quantidade de desenvolvedores.

Aspecto	**Tradicional**	**Utilizando TDD**
Detecção de Erros	Após implementação	Antes da implementação
Qualidade do Código	Pode variar	Consistente
Cobertura de Testes	Geralmente baixa	Alta
Tempo de Desenvolvimento	Inicialmente menor	Compensado a longo prazo

Tabela 2.1: Comparativo entre o desenvolvimento tradicional e o desenvolvimento utilizando TDD.

2.3.3 Arquitetura Cliente-Servidor

A arquitetura cliente-servidor [6] é um modelo bastante utilizado no desenvolvimento de aplicações *Web*. Esse modelo se caracteriza pela separação de responsabilidades entre o cliente, que faz as solicitações, e o servidor, que processa essas requisições e responde

com os dados ou serviços requisitados. Essa divisão proporciona maior modularidade, escalabilidade e flexibilidade.

No contexto dessa arquitetura, o cliente refere-se à interface de usuário que interage diretamente com o sistema. Geralmente, trata-se de uma aplicação executada no navegador do usuário ou em aplicativos móveis. O cliente é responsável por apresentar a interface gráfica, capturar as interações do usuário e enviar solicitações ao servidor por meio de protocolos de rede, como HTTP/HTTPS. Esse modelo permite que o cliente funcione como a ponte entre o usuário final e a lógica de negócios que está no servidor [7].

O servidor, por sua vez, desempenha um papel crucial ao gerenciar as regras de negócio, processar as solicitações recebidas do cliente e acessar os dados armazenados em bancos de dados. Geralmente, o servidor expõe suas funcionalidades por meio de *APIs (Application Programming Interfaces)*, permitindo que diferentes tipos de clientes consumam seus serviços de forma padronizada. A capacidade do servidor de atender a várias requisições simultâneas é fundamental para garantir a eficiência e a disponibilidade do sistema.

A comunicação entre cliente e servidor ocorre principalmente por meio do protocolo HTTP (*Hypertext Transfer Protocol*), utilizado para enviar e receber dados. As requisições HTTP seguem um formato padrão que inclui métodos como GET, POST, PUT e DELETE, permitindo operações de leitura, criação, atualização e exclusão de dados.

Entre as principais vantagens da arquitetura cliente-servidor está a separação de responsabilidades, que facilita o desenvolvimento, a manutenção e a escalabilidade do sistema. Além disso, a modularidade proporcionada por essa arquitetura possibilita a adição de novas funcionalidades sem a necessidade de grandes alterações no sistema já existente.

Outro benefício significativo é a flexibilidade, onde um mesmo servidor pode atender a diferentes tipos de clientes, como navegadores e aplicativos móveis. Onde, por meio de APIs, múltiplos clientes possam consumir os serviços oferecidos de maneira consistente e eficiente.

A segurança também é um aspecto relevante na arquitetura cliente-servidor. Como o servidor centraliza o acesso aos dados e à lógica de negócios, é possível implementar mecanismos robustos de autenticação, autorização e criptografia para proteger as informações sensíveis. Dessa forma, o sistema consegue garantir que apenas usuários autorizados tenham acesso a determinados recursos, aumentando a confiabilidade e a proteção dos dados armazenados.

2.3.4 Mobile First

Mobile First é uma metodologia de desenvolvimento *front-end* que consiste em focar, desde o início do projeto, na arquitetura de telas e componentes voltados para dispositivos

móveis, como *smartphones* e *tablets*, que possuem telas menores e recursos de hardware mais limitados. Essa abordagem implica que a experiência do usuário nesses dispositivos seja priorizada durante o processo de desenvolvimento, garantindo que o site ou aplicativo seja funcional e eficiente mesmo em tamanhos de tela reduzidos.

Esta metodologia vem ganhando cada vez mais espaço no desenvolvimento *Web* devido ao crescimento do número de usuários que acessam a internet por meio de dispositivos móveis. Sendo de grande importância o desenvolvimento de interfaces mais leves e **responsivas**. A partir dessa base, é possível expandir o projeto para telas maiores, como as de *desktops*, garantindo uma experiência fluida em diferentes tamanhos de dispositivos.

2.4 Ferramentas e Tecnologias

Para o desenvolvimento do software, foram utilizadas diversas ferramentas e tecnologias que proporcionaram uma estrutura robusta e escalável. A seguir, apresenta-se uma descrição detalhada das principais tecnologias empregadas:

2.4.1 TypeScript

O **TypeScript** [8] é uma linguagem de programação que estende o *JavaScript*, adicionando suporte à tipagem estática opcional. Diferente do *JavaScript*, que utiliza verificação dinâmica de tipos, o *TypeScript* realiza a verificação estática em tempo de desenvolvimento, permitindo que erros de tipagem sejam detectados antes da execução do código. Isso reduz a ocorrência de falhas inesperadas em tempo de execução e melhora a confiabilidade do software.

Além disso, o *TypeScript* permite o uso de interfaces, tipos explícitos e classes, tornando o código mais organizado, legível e fácil de manter. A tipagem estática também contribui para a documentação do código, pois fornece informações claras sobre as estruturas de dados esperadas. A escolha desta ferramenta foi motivada pela necessidade de um código mais seguro, escalável e sustentável ao longo do tempo, reduzindo riscos associados a erros de tipagem e facilitando a manutenção do sistema.

2.4.2 ReactJS

O **ReactJS** [9] é uma biblioteca *JavaScript* desenvolvida pelo *Facebook* e utilizada para a construção de interfaces de usuário. Seu principal objetivo é facilitar o desenvolvimento de aplicações *Web* dinâmicas e interativas, permitindo que desenvolvedores criem componentes reutilizáveis.

Os principais motivos para a utilização do *React* neste projeto foram a modularidade e o reaproveitamento de código. Sua arquitetura baseada em componentes promove uma organização mais clara e estruturada do código, tornando o desenvolvimento mais ágil, além de uma vasta comunidade e ecossistema de bibliotecas e ferramentas.

A biblioteca adota um paradigma declarativo, permitindo que os desenvolvedores descrevam como a interface deve ser apresentada em diferentes estados, e o *React* cuida de atualizar a interface de forma eficiente quando esses estados mudam.

Uma das principais motivações para a criação do *React* foi melhorar a performance das aplicações *Web*, que sofriam com o re-render desnecessário de elementos da interface. Para resolver esse problema, o *React* introduziu o conceito de Virtual DOM.

Virtual DOM

O Virtual DOM é um dos conceitos centrais do *React*. Trata-se de uma representação em memória da estrutura da interface que permite ao *React* realizar comparações eficientes entre o estado atual e o estado desejado da interface. Quando uma mudança ocorre, o *React* atualiza apenas os elementos necessários no DOM real, reduzindo o tempo de processamento e aumentando a performance [9].

Componentes

O React adota uma abordagem baseada em componentes, que são unidades reutilizáveis de código responsáveis por renderizar uma parte específica da interface. Cada componente pode receber propriedades (props) e gerenciar seu próprio estado interno. Os componentes permitem que a interface seja dividida em partes independentes e reutilizáveis, facilitando a manutenção e a escalabilidade da aplicação.

Outro aspecto importante é o gerenciamento de estado destes componentes. O estado (*state*) refere-se a dados internos do componente que podem mudar ao longo do tempo, enquanto as propriedades (*props*) são valores passados de um componente pai para um componente filho. O estado permite que os componentes do *React* respondam às interações do usuário e mudem dinamicamente a interface conforme necessário [9].

Também é utilizado o conceito de ciclo de vida de componentes, que se refere aos diferentes estágios pelos quais um componente passa, desde sua criação até sua remoção. Esses estágios incluem montagem, atualização e desmontagem. O ciclo de vida dos componentes permite que os desenvolvedores executem operações específicas em momentos distintos, como buscar dados de uma API durante a montagem ou limpar recursos durante a desmontagem [9].

2.4.3 Bootstrap

O **Bootstrap** [10] é um *framework front-end* amplamente utilizado para estilizar interfaces. Ele oferece uma coleção de componentes pré-estilizados e uma grade responsiva que facilita a criação de designs consistentes e adaptáveis a diferentes tamanhos de tela. No projeto, o *Bootstrap* foi fundamental para garantir que a interface fosse criada de forma mais rápida, visualmente agradável e responsiva.

2.4.4 NestJS

O **NestJS** [11] é um *framework* progressivo para o desenvolvimento de aplicações *back-end* em Node.js, que utiliza os conceitos do *TypeScript* para proporcionar uma estrutura sólida e escalável. Inspirado por *frameworks* como Angular, o *NestJS* adota uma abordagem modular, orientada a componentes, utilizando *decorators*, injeção de dependência, e uma arquitetura baseada em módulos, *controllers*, *services* e *guards* de rota, que trabalham em conjunto para fornecer uma base sólida e escalável para aplicações *Web*.

Por ser construído sobre o *Express.js*, o *NestJS* herda a simplicidade e eficiência do ecossistema *JavaScript*, enquanto melhora a experiência do desenvolvedor ao adicionar ferramentas que facilitam a criação de APIs robustas e bem estruturadas. Um dos pilares do *NestJS* é a implementação do paradigma de programação orientado a objetos, juntamente com programação funcional e arquitetura reativa, permitindo que desenvolvedores construam soluções modernas e eficientes.

Outro ponto fundamental do *NestJS* é sua compatibilidade com diversas bibliotecas, como o TypeORM, facilitando a integração com bancos de dados relacionais, além de fornecer suporte nativo a testes automatizados, o que se alinha diretamente ao uso do Test-Driven Development (TDD) na construção de aplicações.

Guards

Os *Guards* no *NestJS* são um dos principais recursos utilizados para implementar controle de acesso e autenticação em rotas da aplicação. Eles funcionam como uma camada intermediária que intercepta a requisição antes que ela seja processada pelo controlador, permitindo que sejam realizadas verificações específicas para determinar se a requisição deve ou não ser autorizada a continuar.

Guards implementam a interface *CanActivate*, que exige a implementação de um método chamado *canActivate()*. Esse método retorna um valor que indica se a requisição pode prosseguir. Dentro desse método, é possível realizar verificações como a presença de um *token* de autenticação, permissões específicas de usuário ou até mesmo validações personalizadas baseadas nos requisitos do negócio.

Além disso, *Guards* podem ser aplicados de forma global ou em rotas específicas, permitindo flexibilidade no controle de acesso. A utilização de *Guards* reforça a segurança e a confiabilidade da aplicação, garantindo que apenas requisições autorizadas sejam processadas.

2.4.5 Swagger

O **Swagger** [12] é uma ferramenta amplamente utilizada para a documentação de APIs REST, proporcionando uma interface interativa que permite visualizar e testar os *endpoints* da aplicação de maneira dinâmica. Por meio de sua especificação o *Swagger* facilita a padronização da documentação, tornando mais simples para desenvolvedores compreenderem e interagirem com a API. Com o uso dessa ferramenta, é possível descrever detalhes como rotas, métodos suportados, parâmetros, respostas esperadas e códigos de status, além de permitir a execução de requisições diretamente pela interface.

Uma das principais vantagens do *Swagger* é sua capacidade de gerar documentação em tempo real a partir do código-fonte, garantindo que a documentação permaneça sincronizada com a API à medida que novas funcionalidades são implementadas. Além disso, ele é amplamente utilizado em processos de desenvolvimento de APIs e integra-se bem com outras práticas, como Test-Driven Development (TDD) [12]. Ferramentas como *Swagger* UI e *Swagger* Editor são frequentemente utilizadas em conjunto para garantir que as APIs estejam bem documentadas e acessíveis para desenvolvedores e consumidores.

2.4.6 Jest

O **Jest** [13] é uma biblioteca de testes em *JavaScript* que permite criar testes unitários, de integração e funcionais. Ele possui funcionalidades como *mocks* automáticos e análise da cobertura de código, que ajudam a garantir a qualidade e a confiabilidade do software. No projeto, o Jest foi utilizado para implementar os testes unitários automatizados conforme a abordagem TDD.

2.4.7 MinIO

O **MinIO** é uma plataforma de armazenamento compatível com o protocolo S3 da Amazon. Foi utilizado para armazenar imagens de maneira segura e eficiente, permitindo que o sistema manipule grandes volumes de dados de forma escalável. Sua simplicidade e alta performance o tornam uma escolha popular para aplicações que exigem armazenamento de arquivos [14].

2.4.8 MySQL

O **MySQL** [15] é um sistema de gerenciamento de banco de dados relacional amplamente utilizado. Foi escolhido para armazenar os dados do sistema de forma estruturada, garantindo integridade e consistência. Sua compatibilidade com diferentes *frameworks* e linguagens de programação, além de sua confiabilidade, o tornam uma escolha ideal para aplicações *Web*.

2.4.9 Google Maps

A **API do Google Maps** [16] foi utilizada para fornecer funcionalidades de geolocalização e mapas interativos dentro do sistema. Com a API, foi possível implementar recursos como cálculo de rotas entre a moradia e a instituição e visualização da localização em mapa. A escolha por essa API se deu pela sua facilidade de uso, documentação abrangente e dados atualizados.

2.4.10 Docker

O **Docker** [17] é uma ferramenta de virtualização baseada em contêineres que facilita a criação, distribuição e execução de aplicações de maneira isolada em diferentes ambientes. Com o Docker, é possível empacotar uma aplicação e as suas dependências em um contêiner leve e portátil, garantindo que o software funcione da mesma forma em qualquer infraestrutura que suporte o Docker. Neste projeto foi utilizado Docker para a execução das aplicações MinIO, MySQL e *back-end* do software.

Com o uso dessas ferramentas e tecnologias, foi possível garantir que o software fosse desenvolvido com alta qualidade, segurança e escalabilidade, atendendo aos requisitos necessários para um software comercial atualmente.

2.4.11 Figma

O **Figma** [18] é uma ferramenta de design de interfaces completamente online, permitindo a prototipação de telas, sem a necessidade de instalar algum software. Essa acessibilidade torna o Figma uma escolha popular entre designers, desenvolvedores e outros profissionais envolvidos na criação de softwares.

Uma das grandes vantagens do Figma é a possibilidade de validar rapidamente ideias e realizar ajustes conforme necessário. Graças aos recursos de prototipagem integrada, os usuários podem criar fluxos de navegação interativos sem precisar recorrer a ferramentas

externas. Isso acelera o processo de desenvolvimento, permitindo refinamentos antes da implementação final.

Outra funcionalidade relevante é a exportação de elementos para uso em código, permitindo que os desenvolvedores obtenham informações precisas sobre espaçamentos, cores e tipografia. Bastante útil em abordagens, onde a consistência visual e a padronização de componentes são essenciais para garantir a qualidade do produto.

A escolha do Figma para este projeto foi motivada pela sua facilidade de prototipação de interfaces de aplicações *Web*.

2.5 Criptografia e Segurança

A criptografia é a prática de proteger informações através da transformação de dados legíveis em um formato cifrado, garantindo sua confidencialidade, integridade e autenticidade. Historicamente, a criptografia tem sido utilizada para proteger comunicações e dados sensíveis, sendo amplamente adotada em sistemas de informação modernos para garantir a segurança digital.

No contexto das aplicações *Web*, a criptografia desempenha um papel fundamental na proteção de dados sensíveis, como senhas, *tokens* de autenticação e informações pessoais. Este tópico apresenta os principais conceitos relacionados à criptografia usados no desenvolvimento do projeto, em especial a autenticação utilizando JSON Web Token (JWT) e o *hash* de senhas.

2.5.1 JSON Web Token

JSON Web Token (JWT) [19] é um padrão aberto baseado em JSON que permite a troca segura de informações entre partes. Um JWT é composto por três partes: *Header* (Cabeçalho), *Payload* (Corpo) e *Signature* (Assinatura), que juntos formam um *token* compacto e auto-suficiente. Esse *token* é amplamente utilizado em sistemas de autenticação e autorização, especialmente em arquiteturas baseadas em APIs.

A estrutura de um JWT é representada pela seguinte composição:

$$\text{JWT} = \text{Header.Payload.Signature} \tag{2.1}$$

- **Header:** Contém informações sobre o tipo de *token* e o algoritmo de assinatura utilizado.
- **Payload:** Inclui as informações (*claims*) que estão sendo transmitidas, como identificação do usuário e tempo de expiração do *token*.

- **Signature:** É gerada a partir da combinação do *header* e do *payload*, utilizando um segredo ou chave privada, garantindo a integridade do *token*.

A utilização de JWT em sistemas *Web* traz benefícios como a redução da necessidade de manter sessões no servidor, maior escalabilidade e interoperabilidade entre diferentes plataformas [20].

2.5.2 Hash de Senhas com Salt

Conceito de Hash

Hashing é uma técnica que transforma uma entrada de dados, como uma senha, em uma sequência fixa de caracteres conhecida como *hash*. O *hash* é irreversível, o que significa que não é possível reconstruir a senha original a partir do *hash* gerado. Essa característica torna o *hashing* uma prática segura para armazenar senhas [21].

Contudo, o uso de *hashing* simples pode ser vulnerável a ataques como dicionário e força bruta. Para mitigar esses riscos, é utilizado o conceito de **salt**, que consiste em adicionar um valor aleatório único a cada senha antes de aplicar a função de *hash*.

Importância do Salt

O uso de *salt* é fundamental para evitar que dois usuários com a mesma senha tenham o mesmo *hash* armazenado no banco de dados. Além disso, o *salt* torna mais difícil a utilização de tabelas pré-computadas de *hashes* de senhas.

O processo de *hashing* com *salt* pode ser descrito pelas seguintes etapas:

1. Geração de um *salt* aleatório.
2. Concatenar a senha do usuário com o *salt*.
3. Aplicação da função de *hash* ao valor concatenado.
4. Armazenamento do *hash* resultante e do *salt* no banco de dados.

2.5.3 Considerações sobre Segurança

A segurança em sistemas de autenticação não depende apenas da implementação de JWT ou hash de senhas com salt, mas de uma abordagem abrangente que considere [22]:

- O uso de protocolos seguros como HTTPS.
- Armazenamento seguro de chaves secretas.
- Monitoramento contínuo de vulnerabilidades e atualizações de bibliotecas utilizadas.

Com a combinação de JWT para autenticação e *hash* de senhas com *salt*, é possível garantir um nível elevado de segurança em aplicações *Web*, protegendo tanto os dados dos usuários quanto os recursos do sistema.

2.6 Lei Geral de Proteção de Dados (LGPD)

A *Lei Geral de Proteção de Dados Pessoais (LGPD)*, instituída pela Lei nº 13.709/2018 [23], entrou em vigor no Brasil com o objetivo de regulamentar o tratamento de dados pessoais, garantindo a proteção da privacidade e dos direitos fundamentais dos cidadãos. Inspirada no Regulamento Geral de Proteção de Dados (GDPR) da União Europeia [24], a LGPD estabelece normas para a coleta, armazenamento, processamento e compartilhamento de dados pessoais, impondo responsabilidades a quem realiza essas atividades.

A LGPD é especialmente relevante para sistemas que realizam o gerenciamento de dados pessoais de usuários, como aplicativos que oferecem autenticação e mantêm bancos de dados com informações sensíveis. A lei define dado pessoal como qualquer informação relacionada a uma pessoa identificada ou identificável, como nome, CPF, e-mail, endereço IP e outros dados que possam levar à identificação do usuário. Além disso, a legislação diferencia os *dados pessoais sensíveis*, que incluem informações sobre origem racial ou étnica, convicções religiosas, dados de saúde e biometria, exigindo um tratamento ainda mais rigoroso.

2.6.1 Impactos da LGPD no Desenvolvimento de Aplicações Web

Em uma aplicação *Web* que utiliza um banco de dados de usuários com funcionalidades de autenticação, a LGPD impõe diversos desafios e obrigações para garantir o cumprimento dos princípios de proteção de dados. Entre os principais pontos a serem considerados no desenvolvimento da aplicação estão:

- **Base Legal para Tratamento de Dados:** O tratamento de dados pessoais deve ser realizado com base em uma das hipóteses legais previstas na legislação. No caso de uma aplicação *Web* que realiza autenticação de usuários, uma base legal comumente utilizada é o consentimento do titular, obtido de forma clara e explícita durante o processo de cadastro [25].
- **Princípios da LGPD:** A aplicação deve seguir os princípios estabelecidos pela LGPD, como:

  - *Finalidade:* O tratamento de dados deve ter uma finalidade específica, informada ao titular no momento da coleta [23].
  - *Adequação:* Os dados coletados devem ser compatíveis com a finalidade informada [23].
  - *Necessidade:* Apenas os dados estritamente necessários para a finalidade proposta devem ser coletados [23].
  - *Segurança:* Medidas técnicas e organizacionais devem ser adotadas para proteger os dados pessoais contra acessos não autorizados, perda, destruição ou vazamento [25].

- **Consentimento e Direitos do Titular:** O consentimento do usuário deve ser obtido de forma explícita, destacando quais dados estão sendo coletados, a finalidade do uso e os direitos do titular. A LGPD garante aos usuários direitos como:
  - Direito de acesso aos dados armazenados [23];
  - Direito de correção de dados incompletos, inexatos ou desatualizados [23];
  - Direito de eliminação de dados pessoais, salvo em situações em que a lei exige o armazenamento [23];
  - Direito à portabilidade dos dados para outro fornecedor de serviço [23];
  - Direito à revogação do consentimento a qualquer momento [23].

2.6.2 Medidas de Segurança e Boas Práticas

O desenvolvimento de aplicações *Web* que manipulam dados pessoais deve incorporar medidas de segurança que minimizem riscos de vazamentos ou acessos indevidos. Entre as práticas recomendadas estão:

- Implementação de mecanismos de autenticação seguros [25];
- Limitação do acesso aos dados pessoais a usuários e sistemas que realmente necessitam dessas informações [25];
- *Logs* de auditoria para monitorar e registrar ações realizadas sobre os dados pessoais [25];

2.6.3 Conformidade e Riscos Jurídicos

Garantir a conformidade com a LGPD é essencial para evitar incidentes relacionados ao tratamento inadequado de dados pessoais. Isso implica que a proteção dos dados pessoais

não deve ser tratada como uma etapa final do desenvolvimento, mas como um aspecto central de todo o ciclo de vida do software. Essa abordagem é conhecida como *privacy by design*, onde a privacidade é incorporada desde o início do projeto [24].

Capítulo 3

Desenvolvimento

Este capítulo aborda as etapas fundamentais do desenvolvimento do projeto, desde a análise de requisitos até a apresentação dos resultados obtidos. Serão detalhados os processos de prototipação, configurações iniciais, modelagem do banco de dados, implementação do servidor e do cliente, com o objetivo de fornecer uma visão clara e estruturada de como a solução foi construída e validada.

3.1 Análise de Requisitos

A análise de requisitos é o processo de descrever como o software deve se comportar, listando funcionalidades que devem compor a aplicação.

3.1.1 Introdução

O projeto tem como principal objetivo conectar estudantes que procuram vagas em repúblicas estudantis com proprietários ou responsáveis pelas vagas disponíveis. Como discutido anteriormente na seção 2.2 Ferramentas Semelhantes, a aplicação a ser desenvolvida deve ser especializada na busca de repúblicas estudantis, oferecendo uma experiência de usuário intuitiva e eficiente. Além disso, a plataforma deve atender as principais necessidades dos estudantes, como a busca por acomodações próximas às instituições de ensino, informações detalhadas sobre as moradias e opções de transporte público, superando as limitações identificadas nas soluções existentes. O sistema permitirá o cadastro de anúncios com informações detalhadas sobre a república, incluindo:

- Nome do anúncio
- Descrição das condições de moradia
- Endereço completo

- Localização no mapa
- Fotos dos ambientes
- Valor mensal em reais
- Contato do responsável

3.1.2 Requisitos Funcionais

Os requisitos funcionais estabelecem as capacidades e comportamentos essenciais que a aplicação deve oferecer aos usuários. Eles abrangem as principais operações do negócio, desde o gerenciamento de entidades básicas até processos mais complexos.

Cadastro de Usuários

- **RF01:** O sistema deve permitir o cadastro de usuários, com nome completo, um e-mail válido, número de telefone e senha.
- **RF02:** O sistema deve permitir o login e autenticação de usuários por e-mail e senha.

Cadastro de Anúncios

- **RF03:** O sistema deve permitir que usuários cadastrem anúncios.
- **RF04:** Cada anúncio deve conter informações como endereço, descrição, valor mensal, gêneros permitidos e fotos do imóvel.
- **RF05:** O sistema deve permitir a inserção de até 3 fotos por anúncio.
- **RF06:** No cadastro de anúncio deve permitir, opcionalmente, a associação de uma instituição de ensino pré cadastrada.

Busca e Filtro de Anúncios

- **RF07:** O sistema deve permitir que os estudantes filtrem os anúncios por estado, gênero, tipo de acomodação, instituição e título.
- **RF08:** O sistema deve exibir os anúncios em ordem de de criação e de forma paginada.
- **RF09:** O sistema deve exibir os anúncios em formato de lista, com fotos e informações principais.

- **RF10:** Quando selecionado, por um usuário, um anúncio da lista deve ser levado para a página de detalhes do anúncio da acomodação.

Visualização de Anúncios

- **RF11:** O sistema deve exibir informações detalhadas do anúncio.
- **RF12:** O sistema deve exibir a localização do imóvel em um mapa interativo.
- **RF13:** Se o anúncio possuir uma instituição associada, deve mostrar a rota e o tempo de deslocamento até a instituição.
- **RF14:** O sistema deve exibir informações de contato do anunciante e a localização exata apenas para usuários autenticados.

Edição de anúncios

- **RF15:** O sistema deve permitir que anunciantes editem informações do anúncio.
- **RF16:** O sistema deve permitir remover e adicionar fotos a um anúncio.
- **RF17:** O sistema deve permitir excluir um anúncio.

3.1.3 Requisitos Não Funcionais

Os requisitos não funcionais definem os critérios de qualidade, desempenho e aspectos técnicos que o sistema deve atender. Eles englobam aspectos como segurança e usabilidade, estabelecendo padrões para a arquitetura da aplicação, infraestrutura e experiência do usuário. Esses requisitos são essenciais para garantir que o sistema seja eficiente e robusto, complementando as necessidades funcionais com atributos que impactam diretamente sua confiabilidade e manutenibilidade.

- **RNF01:** O sistema deve ser acessível via *desktop* e dispositivos móveis (responsivo).
- **RNF02:** O sistema deve ter uma interface para documentar e interagir com o *back-end*.
- **RNF03:** As imagens enviadas pelos usuários devem possuir no máximo 200 kB, para ocupar menos armazenamento e garantir um carregamento rápido.
- **RNF04:** Ao excluir um anúncio ou usuário deve ser mantido o registro.

Requisitos de Segurança

- **RNF05:** As senhas devem ser armazenadas de forma segura e criptografada.
- **RNF06:** O sistema deve prevenir ataques de injeção de SQL.
- **RNF07:** O sistema deve utilizar uma interface segura para armazenar informações de usuários e anúncios.
- **RNF08:** O sistema não deve permitir alterações de anúncios de terceiros.

A análise de requisitos tem como objetivo garantir que o software atenda às necessidades básicas dos usuários, permitindo que estudantes encontrem repúblicas estudantis de maneira rápida, segura e eficiente. A implementação dos requisitos funcionais e não funcionais, aliada ao uso de tecnologias eficientes, garantirá a entrega de uma aplicação de qualidade e com boa experiência de usuário.

3.2 Prototipação de telas

O desenvolvimento de um projeto de software requer um planejamento cuidadoso e estruturado, sendo a prototipação de telas outra etapa bastante importante para garantir a qualidade do software final. A prototipação consiste em criar modelos visuais das telas do sistema antes do início da codificação, possibilitando uma visão geral de como será a interação do usuário com a aplicação. Essa prática auxilia na identificação de fluxos de navegação, funcionalidades essenciais e elementos visuais que devem ser implementados, além de permitir ajustes antecipados antes que o desenvolvimento avance para etapas mais complexas.

Durante a fase de prototipação, foi possível mapear as principais entidades necessárias no banco de dados, pois ao visualizar as telas, é possível identificar quais informações precisarão ser armazenadas e como essas informações estarão interligadas. Esse mapeamento inicial é fundamental para a construção de um banco de dados eficiente e bem estruturado, reduzindo a chance de retrabalho e problemas de performance no futuro. Além disso, a prototipação contribuiu para o planejamento das tarefas que deverão ser realizadas na etapa de desenvolvimento. Cada tela ou funcionalidade identificada no protótipo pode ser transformada em uma tarefa específica, facilitando a organização e a gestão do projeto.

Outro benefício significativo da prototipação é a possibilidade de analisar quais tecnologias serão mais adequadas para o desenvolvimento do projeto. Com base nas funcionalidades previstas e na complexidade do sistema, é possível definir quais *frameworks*, bibliotecas e linguagens de programação serão mais eficientes para atingir os objetivos do

projeto. Essa análise tecnológica, quando realizada de forma antecipada, evita a escolha de ferramentas inadequadas que possam comprometer o andamento do desenvolvimento.

3.2.1 Fluxograma

A etapa seguinte foi criar um fluxograma (Figura 3.1) mapeando a navegação no sistema, desde a tela inicial até funcionalidades, como a busca de acomodações, autenticação e cadastro de acomodações. Utilizando o LucidChart, foi possível a modelagem do fluxograma de forma interativa, online e gratuita.

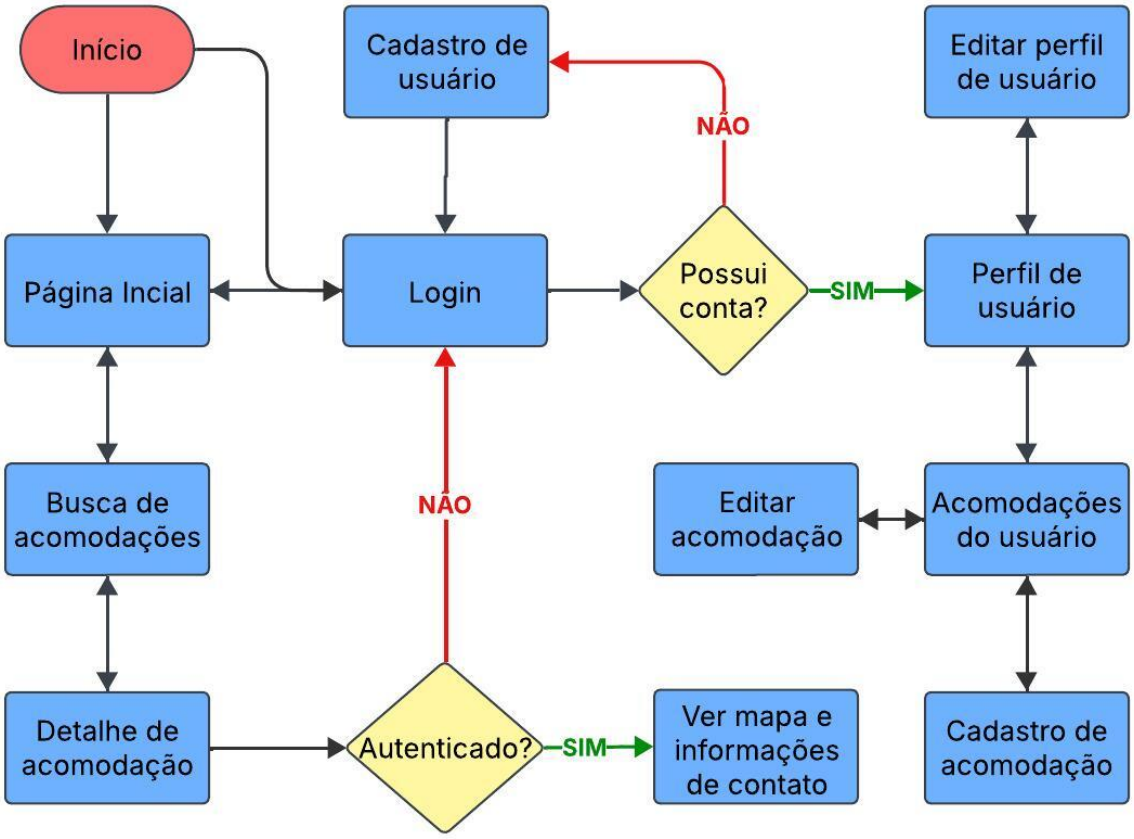


Figura 3.1: Fluxograma de navegação - LucidChar.

3.2.2 Protótipos

Para a prototipação deste projeto foi utilizado o **Figma** [18], uma ferramenta amplamente reconhecida por sua eficiência na criação de interfaces visuais. Permitindo a validação rápida das ideias e realização dos ajustes necessários.

Todas as páginas foram prototipadas com um cabeçalho que se estende por toda largura do dispositivo, com o nome do software e um menu de opções. O resultado desta etapa foi a prototipação das 6 seguintes telas com as respectivas finalidades:

Tela inicial

Tela com o objetivo de receber usuários com uma apresentação breve do projeto e com alguns *links* rápidos para busca de acomodações por instituição ou estado. Figura 3.2

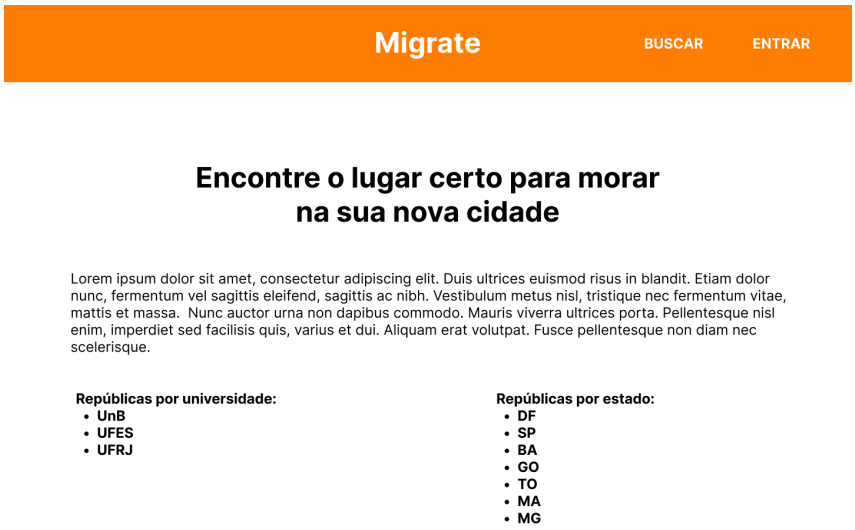


Figura 3.2: Protótipo da tela inicial.

Tela de cadastro de usuário

Tela com o objetivo de realizar o cadastro de novo usuário no sistema, solicitando a entrada dos dados: nome, e-mail, telefone, senha e confirmação de senha. Figura 3.3

Tela de login

Tela para realização da autenticação de um usuário já cadastrado, através de e-mail e senha. Possuindo *link* para usuários não cadastrados “Criar Conta”. Figura 3.4

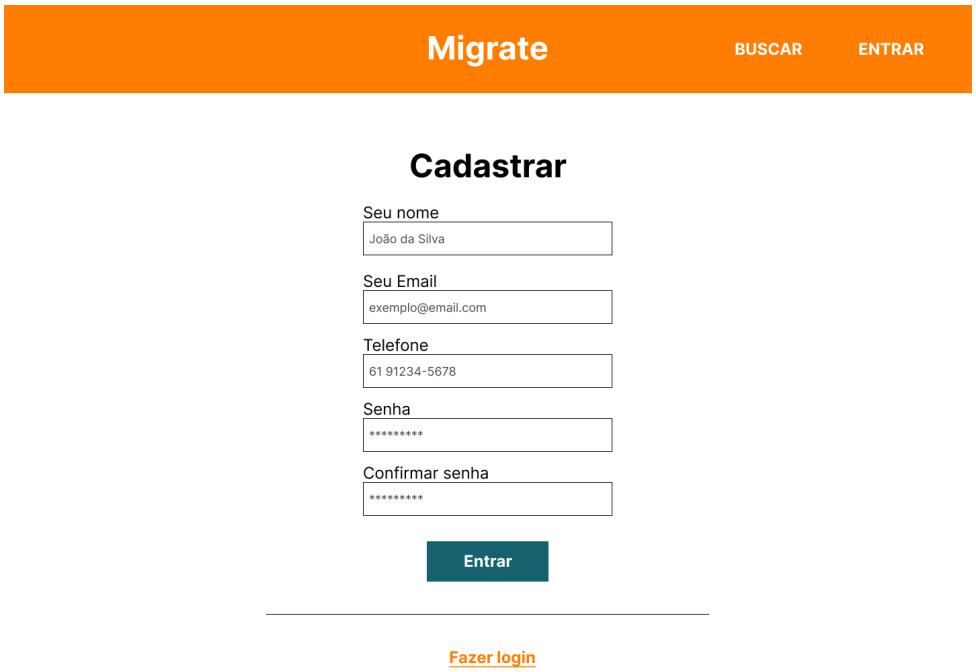


Figura 3.3: Protótipo da tela de cadastro de usuário.

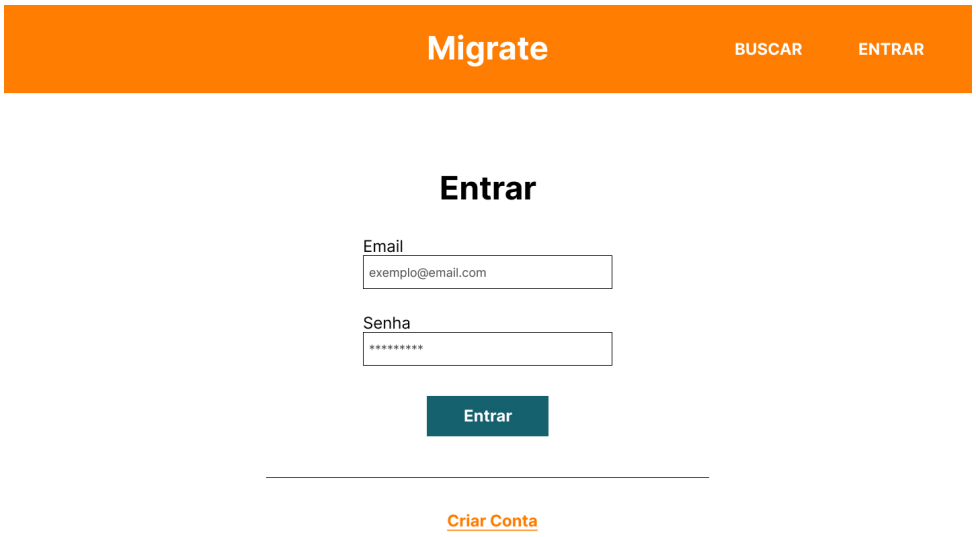


Figura 3.4: Protótipo da tela de login.

Tela de cadastro de acomodação

Tela com formulário de cadastro de acomodação, solicitando a entrada dos dados: título, valor, descrição, endereço, instituição, gêneros aceitos e fotos. Esta tela é acessível apenas para usuários autenticados (Figura 3.5).



Figura 3.5: Protótipo de tela de cadastro de acomodação.

Tela de exibição de acomodação

Tela que mostra dados de um anúncio de acomodação, com um carrosel das fotos, título, valor, descrição, endereço, instituição, gêneros aceitos, informações de contato e mapa dinâmico com a localização e rota até a instituição associada. Vale ressaltar que informações de contato e mapa o usuário deve estar autenticado para visualizar, para proteger, minimamente, informações sensíveis dos usuários, contra bots e diminuir a quantidade de consultas de mapa da API do Google, visto que é uma API gratuita de uso limitado. Figura 3.6

Tela de busca de acomodação

Tela com cartões dos anúncios de acomodações, exibidos de forma paginada e com opções de filtragem para uma busca mais precisa. Filtros por título, universidade, estado, gênero e/ou tipo de acomodação. Ao clicar em um cartão é levado a tela de exibição de acomodação. Figura 3.6

A visualização dos protótipos de tela possibilitou uma compreensão mais ampla e detalhada da complexidade do projeto. Neste processo foi possível identificar os principais desafios envolvidos, bem como para selecionar as ferramentas e tecnologias mais adequadas para o desenvolvimento.

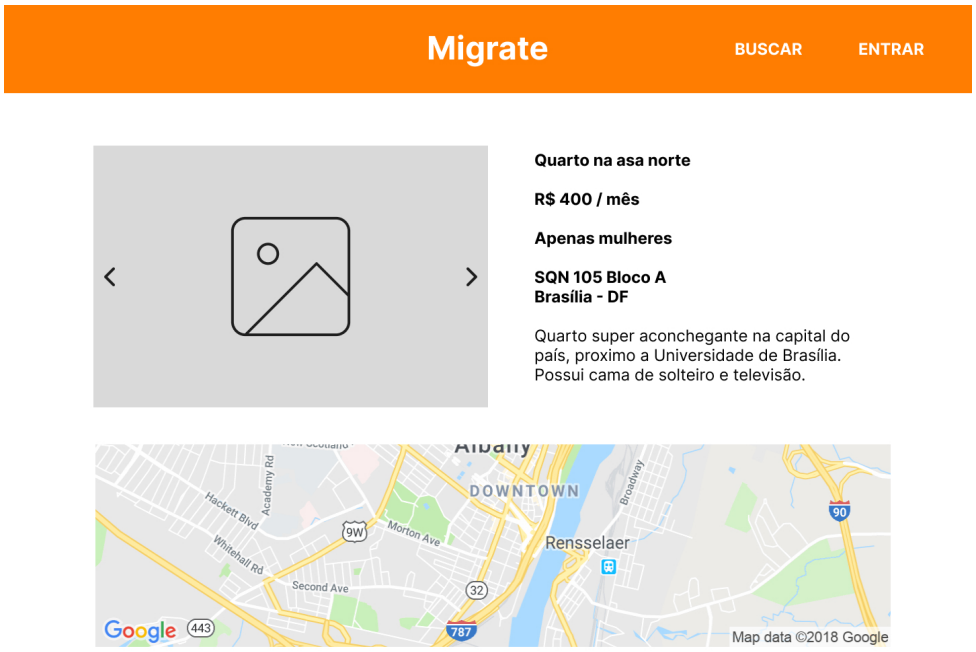


Figura 3.6: Protótipo de tela de acomodação.

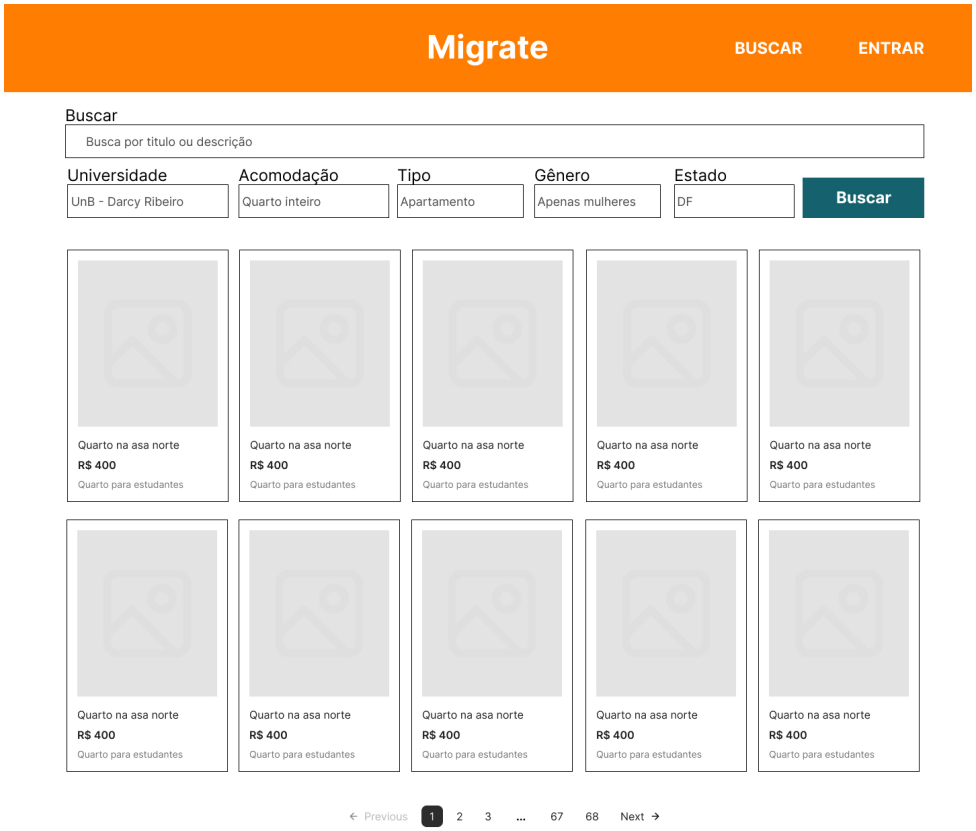


Figura 3.7: Protótipo de tela de busca de acomodação.

3.3 Configurações Iniciais

Visto o protótipo de interface e tendo um bom entendimento das regras de negócio o próximo passo foi criar e organizar as tarefas iniciais no *backlog* do quadro *Kanban*, categorizando por tipo e prioridade.

Sendo três categorias de prioridade: baixa, média e alta, e três tipos de tarefas:

Tipo	**Descrição**
Infraestrutura	Tarefas de configuração de máquina, serviços apartados e publicações (deploys)
Back-end	Tarefas de código no projeto back-end
Front-end	Tarefas de código no projeto front-end

Tabela 3.1: Tipos de tarefas do quadro Kanban

Em seguida foram iniciadas as tarefas pela ordem de prioridade:

1. **Configuração do NodeJS**: Antes de iniciar qualquer projeto *Javascript* é necessária a configuração correta do interpretador no sistema, para este projeto foi escolhida sua versão 20.9.0, incluindo também seus gerenciadores de pacotes: *npm* e *npx*
2. **Criação do projeto NestJS**: Foi feita a instalação do pacote do *framework* NestJS CLI [11] no sistema e em seguida seguidos os passos da documentação
3. **Criação do projeto ReactJS com Typescript**: Seguida a documentação do *software Create React App* [26] e configurando para utilizar o *superset*
4. **Configuração de arquivo Docker Compose**: Tendo em vista a complexidade do projeto e pensando na simplificação de publicação do software, removendo a necessidade de configurar cada ambiente manualmente, foi feita a configuração do banco de dados **MySQL** e da plataforma de armazenamento **MinIO** em um arquivo "docker-compose.yaml". Usando imagens *Docker* fornecidas pelos mantenedores de cada uma das ferramentas. Isso resultou em maior eficiência no processo de desenvolvimento e no controle de configurações dos ambientes utilizados.

Agora com as ferramentas necessárias foram iniciadas tarefas de codificação para moldar o software de acordo com as regras de negócio.

3.4 Banco de Dados

O uso de banco de dados é essencial para armazenar e gerenciar dados de forma estruturada, garantindo integridade, consistência e consultas eficientes. A escolha de um modelo relacional deve-se à necessidade de relacionamentos bem definidos entre entidades (como usuários, anúncios e instituições), garantindo normalização e evitando redundâncias. Neste capítulo será abordada a modelagem e implementação do banco de dados relacional, com o objetivo de configurar um banco de dados eficiente, escalável e alinhado às regras de negócio do sistema.

3.4.1 Modelagem

A próxima etapa do projeto foi a modelagem do banco de dados, pensando quais entidades seriam necessárias, seus atributos e relacionamentos. A modelagem de dados foi uma etapa fundamental no desenvolvimento, representando de forma organizada e estruturada as informações armazenadas no banco de dados. As chaves primárias das entidades são do tipo Universally unique identifier (UUID).

Usuário

Entidade que representa os usuários do sistema e armazena informações essenciais para a identificação e autenticação. Cada usuário possui os seguintes atributos:

- Identificador único;
- Nome completo;
- E-mail, utilizado para autenticação e contato, não pode ser repetido;
- Senha, criptografada, utilizada para autenticação [21];
- *Salt* da senha armazenada [21];
- Número de telefone de contato do usuário;
- Gênero informado pelo usuário, do tipo textual;
- Data e hora de criação do registro;
- Data e hora de exclusão lógica do registro.

Localização

Entidade que representa os endereços, de forma precisa, de acomodações ou instituições. Cada localização possui os seguintes atributos:

- Identificador único;
- Código de Endereçamento Postal (CEP);
- Nome do país;
- Nome do estado;
- Nome da cidade;
- Nome do bairro;
- Complemento do endereço;
- Latitude, fornecida pela API do Google Maps;
- Longitude, fornecido pela API do Google Maps.

Instituição

Entidade que representa as instituições de ensino cadastradas no sistema, servindo para a filtragem de anúncios. Cada instituição possui os seguintes atributos:

- Identificador único;
- Nome da instituição;
- **Localização** como chave secundária.

Acomodação

Entidade que representa os anúncios de acomodações no sistema, caracterizadas com os seguintes atributos:

- Identificador único;
- Título do anúncio;
- Descrição do anúncio;
- **Usuário** criador do anúncio;
- **Localização** como chave secundária;
- **Instituição**: Pode-se, opcionalmente, associar uma instituição de ensino ao anúncio de acomodação. Essa associação serviria para estudantes que tem preferência por conviver com estudantes de sua mesma instituição;
- Preço mensal do aluguel em reais;

- Gêneros aceitos na acomodação, podendo aceitar apenas homens, apenas mulheres ou qualquer gênero;
- Tipo de contato: Identifica quais informações de contato do usuário que criou o anúncio devem ser mostradas (somente e-mail, somente telefone ou ambos);
- Tipo de acomodação: Identifica se o anúncio de aluguel trata-se do imóvel inteiro, apenas um quarto ou apenas uma cama;
- Tipo do imóvel, podendo ser apartamento, casa ou outro.
- Situação do anúncio, manipulado por administradores do sistema, podendo estar ativo, em análise ou bloqueado;
- Distância, em quilômetros, da instituição de ensino, se associada ao anúncio
- Modo mais comum para ir até a instituição de ensino, se associada ao anúncio. Atributo utilizado pela API do Google Maps para traçar as rotas e estimar o tempo de locomoção.
- Data e hora de criação do registro;
- Data e hora de exclusão lógica do registro.

Foto

Entidade que representa as fotos de acomodações no sistema, possuindo os seguintes atributos:

- Identificador único;
- **Acomodação** a qual a foto pertence;
- URL da foto, armazenada em um serviço apartado (MinIO);
- Ordem de exibição da foto no carrossel do anúncio;
- Data e hora de criação do registro.

3.4.2 Normalização do Banco de Dados

A normalização do banco de dados foi aplicada para garantir a integridade dos dados, evitar redundâncias e facilitar a manutenção do sistema. O processo de normalização seguiu as formas normais até a terceira forma normal (3FN), conforme descrito a seguir:

Primeira Forma Normal (1FN)

Todas as entidades foram projetadas para conter apenas valores atômicos, ou seja, cada atributo armazena um único valor. Por exemplo, na entidade "Localização", o atributo "Tipo de acomodação" foi modelado para evitar múltiplos valores em um único campo.

Segunda Forma Normal (2FN)

Todas as entidades foram verificadas para garantir que todos os atributos não-chave dependam totalmente da chave primária. Por exemplo, na entidade "Foto", todos os atributos dependem exclusivamente do identificador único da foto, e não de outros atributos.

Terceira Forma Normal (3FN)

Foram eliminadas dependências transitivas, garantindo que nenhum atributo não-chave dependa de outro atributo não-chave. Por exemplo, na entidade "Localização", os atributos como "Nome do país", "Nome do estado" e "Nome da cidade" dependem diretamente do identificador único da localização, e não de outros atributos.

Benefícios da Normalização

- **Redução de Redundâncias**: A normalização eliminou a duplicação de dados, como no caso de informações de localização repetidas em múltiplas entidades.
- **Integridade dos Dados**: A aplicação de chaves primárias e estrangeiras garantiu a consistência dos relacionamentos entre as entidades.
- **Facilidade de Manutenção**: A estrutura normalizada permite que alterações no banco de dados sejam realizadas de forma mais segura e eficiente.

3.4.3 Diagrama Entidade-Relacionamento (ER)

Para visualizar de forma clara a estrutura do banco de dados e os relacionamentos entre as entidades, foi desenvolvido um diagrama Entidade-Relacionamento (ER) utilizando o MySQL Workbench. O diagrama ilustra as entidades principais, como "Usuário", "Localização", "Instituição", "Acomodação" e "Foto", além de representar os relacionamentos entre elas, como chaves primárias, chaves estrangeiras e os tipos dos atributos. Figura 3.8

3.4.4 Implementação

A implementação do banco de dados teve início por meio de código, visando maior agilidade no processo de desenvolvimento e reaproveitamento dos arquivos gerados para o

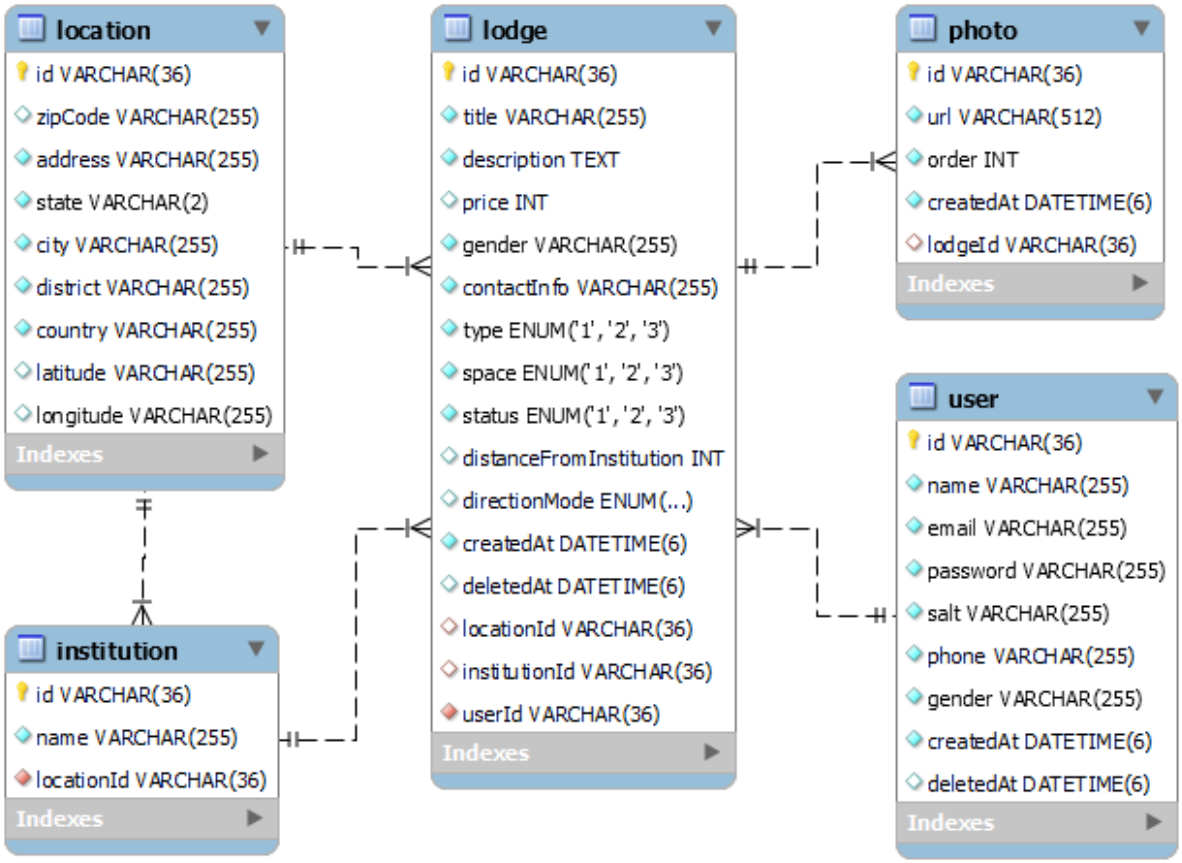


Figura 3.8: Diagrama de entidade-relacionamento dos dados - MySQL Workbench.

uso do serviço *back-end*. Essa abordagem permitiu uma maior integração entre o banco de dados e o código da aplicação, garantindo que as mudanças nas estruturas de dados fossem refletidas diretamente na lógica de negócio.

Para a realização desta etapa, utilizando a biblioteca *TypeORM*, foi necessário criar arquivos de configuração e marcação para cada entidade, utilizando a extensão ".entity.ts". Cada entidade foi definida com base em classes do *TypeScript* e anotada com os *decorators* fornecidos pelo *TypeORM*. Esses *decorators* permitem a especificação de propriedades como colunas, tipos de dados, chaves primárias e relacionamentos entre as tabelas. Garantindo que a estrutura do banco de dados estivesse alinhada com as regras de negócio da aplicação.

O uso do *TypeORM* facilitou a manipulação de dados, possibilitando a sincronização da modelagem no código com a do banco de dados. A implementação do banco de dados utilizando o *TypeORM* no contexto do *NestJS* demonstrou ser uma escolha eficiente e ágil. A utilização de uma abordagem baseada em código possibilitou maior controle sobre as entidades, facilitou a manutenção e garantiu que as evoluções no banco de dados fossem acompanhadas pelo código de forma transparente e automatizada.

Exemplo de implementação

A Figura 3.9 ilustra a implementação de uma entidade no *framework NestJS* utilizando *TypeORM*, por exemplo a entidade "Instituição". A estrutura é definida por meio de uma classe *TypeScript*, onde cada atributo da classe corresponde a uma coluna no banco de dados, como **id, name e location**. Para mapear esses atributos às colunas do banco de dados e definir seus tipos, são utilizados *decorators* do *TypeORM*. O atributo *id* é marcado com **@PrimaryGeneratedColumn** para indicar que se trata de uma chave primária gerada automaticamente, enquanto **name** utiliza **@Column** para definir uma coluna comum. Já o atributo **location**, que representa um relacionamento do tipo um-para-um (OneToOne) com a entidade "Localização", é configurado com os *decorators* **@OneToOne e @JoinColumn**, garantindo a integridade referencial e o mapeamento correto entre as tabelas.



```
@Entity()
export class Institution {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column({ nullable: false })
  name: string;

  @OneToOne(() => Location, { nullable: false, cascade: true })
  @JoinColumn()
  location: Relation<Location>;
}
```

Figura 3.9: Exemplo de código de criação de entidade com TypeORM.

3.5 Implementação Servidor

Nesta seção, será apresentada a implementação do *back-end* da aplicação, utilizando o *framework NestJS* para construção do servidor e a biblioteca Jest para realização de testes automatizados. Foi aplicada a metodologia de Desenvolvimento Orientado por Testes (TDD), que busca garantir a qualidade e confiabilidade do código desde as primeiras etapas do projeto.

3.5.1 Módulos

Seguindo o padrão do *framework* escolhido foram criados 5 (cinco) módulos, cada um com seus respectivos códigos de *controller, service* e DTO, além dos respectivos testes unitários.

Módulos criados:

1. **Auth:** Responsável por funcionalidades de autenticação como cadastro e login. Além da geração do token JWT no login e validação em funcionalidades protegidas.
2. **User:** Funcionalidades protegidas de obtenção dos dados do usuário logado, atualização dos dados e exclusão de conta do usuário.
3. **Institution:** Responsável pelo gerenciamento de instituições de ensino. Funcionalidades de criação, leitura, edição e exclusão de uma instituição de ensino. Além de uma busca geral e busca por estado.
4. **Lodge:** Responsável pelo gerenciamento de anúncios de acomodações. Funcionalidades públicas para busca paginada e visualização de detalhes de um anúncio. E funcionalidades protegidas para criação, edição, exclusão e dados de contato de um anúncio, além da busca dos anúncios cadastrados pelo usuário logado.
5. **Photo:** Responsável pelo gerenciamento de fotos. Funcionalidades de adição ou remoção de fotos de um anúncio.

Para o funcionamento correto do sistema cada um desses módulos devem ser injetados como importações no módulo principal da aplicação (*AppModule*) Figura 3.10, bem como os módulos de configuração do *NestJS* e de configuração do *TypeORM*. Vale observar a configuração das entidades do banco de dados e da *flag* de sincronização ativada, desta forma qualquer alteração feita nos arquivos de código das entidades refletirá no banco de dados.



Figura 3.10: Código do módulo principal *NestJS* com devidas injeções de dependências.

Assinatura e validação de *tokens*

Uma funcionalidade essencial para a segurança da aplicação é a manipulação correta dos *tokens* de autenticação. Ao realizar um login o usuário recebe um *token* assinado com o a chave segredo e quando feita uma requisição a uma rota protegida o *token* é verificado se é válido, ou seja, foi assinado pela chave segredo e sua validade não está expirada.

A implementação da assinatura de *tokens* foi possível através da utilização da biblioteca '*@nestjs/jwt*', que funciona de forma integrada com o servidor fornecendo um serviço de *tokens* que pode ser importado em módulos *NestJS*.

A Figura 3.11 ilustra como foi feita a configuração do módulo JWT. Sendo registrado com as configurações de escopo global, *secret* de assinatura e tempo de expiração dos *tokens*.

```
JwtModule.register({
      global: true,
      secret: process.env.JWT_SECRET,
      signOptions: { expiresIn: process.env.JWT_TIMEOUT || '2h' },
    }),
}
```

Figura 3.11: Código de configuração do módulo NestJS de JSON Web Token.

Já a validação de *tokens* foi implementada utilizando o conceito de *middleware*, através de *Guards NestJS*. Extraindo o *token* da requisição, antes da execução da funcionalidade, validando com o mesmo serviço de *tokens* (Figura 3.13) e lançado um erro de acesso negado caso o *token* não seja válido.

3.5.2 Testes Unitários

Como citado anteriormente foi utilizada a metodologia de Desenvolvimento Orientado por Testes (TDD), utilizando a biblioteca Jest para criação dos testes unitários e em seguida feita a refatoração do código para corrigir os erros apontados.

Cada *controller, service e guard* do projeto possui um arquivo dedicado para testes unitários. Esses arquivos especificam os resultados esperados das chamadas de métodos e das alterações de atributos, além de verificarem o comportamento do código, como a contagem de iterações e o fluxo de execução em diferentes cenários.

A Figura 3.14 demonstra a codificação de um teste unitário utilizando o Jest. Neste exemplo, o método de busca de acomodações é testado em dois cenários extremos: uma busca sem parâmetros e outra com todos os parâmetros preenchidos. Em ambos os casos,

espera-se que a aplicação retorne os resultados fornecidos pelo banco de dados, que foi simulado (*mockado*) para garantir a independência do teste unitário do método.

A aplicação de testes unitários estão diretamente relacionadas ao objetivo de alcançar uma cobertura de testes consistente ao longo do projeto. Cada arquivo de teste foi projetado para validar o comportamento esperado das funções principais, garantindo que tanto os casos de uso comuns quanto as situações excepcionais fossem testadas. O monitoramento da cobertura de testes permitiu identificar possíveis lacunas no processo de validação, direcionando o desenvolvimento de novos testes e a refatoração do código. Com isso, foi possível assegurar maior confiabilidade e robustez ao sistema, alinhando a qualidade do software aos princípios estabelecidos pela metodologia adotada.

A cobertura de testes é um indicador importante que mede a quantidade de código que foi efetivamente testada pelos casos de teste desenvolvidos. Essa métrica permite avaliar quais partes do sistema estão devidamente validadas e identificar trechos que ainda não foram verificados. No projeto, a cobertura de testes foi monitorada para garantir que os principais componentes — como *controllers, services e guards* — fossem testados em diferentes cenários.

O relatório gerado pelo Jest (Figura 3.15) demonstrou que mais de 99% das linhas de código do projeto estão cobertas por testes, o que representa uma cobertura excepcional e reforça a qualidade do software desenvolvido. Esse índice elevado de cobertura é um forte

```
async signIn(email: string, pass: string): Promise<SignInResponse> {
    const user = await this.userService.findOneByEmail(email);
    if (!user) {
      throw new UnauthorizedException();
    }

    const isMatch = await bcrypt.compare(pass, user?.password);
    if (!isMatch) {
      throw new UnauthorizedException();
    }

    const payload: JwtData = {
      sub: user.id,
      id: user.id,
      email: user.email,
      name: user.name,
    };
    return {
      access_token: await this.jwtService.signAsync(payload),
    };
  }
```

Figura 3.12: Código do método de login do módulo de autenticação.

indicador de que o comportamento das funcionalidades foi amplamente validado, incluindo diferentes cenários de uso, entradas variadas e tratamentos de exceções. Além de fornecer confiança sobre a estabilidade do sistema, essa métrica também contribui para a prevenção de regressões, visto que alterações futuras no código serão rapidamente identificadas caso comprometam algum requisito já testado. Assim, a alta cobertura não apenas reflete o compromisso com boas práticas de desenvolvimento, como também assegura que o sistema está mais preparado para enfrentar situações imprevistas em produção, elevando a confiabilidade e a maturidade do projeto como um todo.

```typescript
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(
    private jwtService: JwtService,
    private reflector: Reflector,
  ) {}

  async canActivate(context: ExecutionContext): Promise<boolean> {
    const isPublic = this.reflector.getAllAndOverride<boolean>(IS_PUBLIC_KEY, [
      context.getHandler(),
      context.getClass(),
    ]);
    if (isPublic) {
      return true;
    }

    const request = context.switchToHttp().getRequest();
    const token = this.extractTokenFromHeader(request);
    if (!token) {
      throw new UnauthorizedException();
    }
    try {
      const payload = await this.jwtService.verifyAsync(token, {
        secret: process.env.JWT_SECRET,
      });
      // Access it in route handlers
      request['user'] = payload;
    } catch {
      throw new UnauthorizedException();
    }
    return true;
  }

  private extractTokenFromHeader(request: Request): string | undefined {
    const [type, token] = request.headers.authorization?.split(' ') ?? [];
    return type === 'Bearer' ? token : undefined;
  }
}
```

Figura 3.13: Código do middleware de validação de JSON Web Token.

```
describe('search', () => {
    it('should search without params', async () => {
      mockLodgeRepository.findAndCount.mockResolvedValue([[], 0]);
      expect(await service.search({})).toBeDefined();
    });

    it('should search with all params', async () => {
      mockLodgeRepository.findAndCount.mockResolvedValue([[], 0]);
      const searchParams: SearchLodgeDto = {
        gender: 'any',
        institutionId: '123',
        page: 2,
        space: SpaceType.HOUSE,
        state: 'DF',
        type: LodgeType.ENTIRE,
      };
      expect(await service.search(searchParams)).toBeDefined();
    });
  });
```

Figura 3.14: Código de teste unitário do método de busca de acomodações.

All files

99.15% Statements 353/356 **95.91%** Branches 47/49 **98.66%** Functions 74/75 **99.05%** Lines 313/316

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
src	100%	7/7	100%	0/0	100%	2/2	100%	5/5
src/modules/auth	97.1%	67/69	100%	9/9	90.9%	10/11	96.72%	59/61
src/modules/institution	100%	45/45	100%	0/0	100%	15/15	100%	41/41
src/modules/lodge	100%	100/100	96%	24/25	100%	18/18	100%	87/87
src/modules/photo	97.87%	46/47	100%	0/0	100%	11/11	97.56%	40/41
src/modules/user	100%	33/33	100%	0/0	100%	11/11	100%	29/29
src/utils/file	100%	12/12	50%	1/2	100%	2/2	100%	12/12
src/utils/getCoordinates	100%	25/25	100%	9/9	100%	2/2	100%	22/22
src/utils/latLngDistance	100%	13/13	100%	0/0	100%	2/2	100%	13/13
src/utils/paginate	100%	5/5	100%	4/4	100%	1/1	100%	5/5

Figura 3.15: Relatório gerado pelo Jest da cobertura de teste do projeto.

3.6 Implementação Cliente

Nesta seção, serão apresentados os principais aspectos relacionados à implementação do *front-end* do sistema, desenvolvido com o *framework ReactJS*. Serão abordados temas como a estruturação do código-fonte, o gerenciamento de componentes, e a configuração do roteamento de páginas. Além disso, serão discutidas as práticas utilizadas para garantir uma interface dinâmica e responsiva, destacando a integração com a API do *back-end* e o uso de bibliotecas complementares que otimizaram o desenvolvimento.

3.6.1 Estrutura do Código

A definição da estrutura de pastas do código foi orientada pelos princípios de modularização, reutilização de componentes e simplificação da manutenção do sistema, visando organizar o projeto de forma clara e escalável. Sendo criadas as seguintes pastas na raiz do projeto:

- Assets: Recursos utilizados pelo sistema de forma complementar, como fontes de texto, imagens e ícones.
- Components: Conjunto de componentes reutilizáveis da interface, responsáveis por estruturar e renderizar elementos visuais do sistema, visando modularidade e reaproveitamento de código.
- Contexts: Arquivos responsáveis por gerenciar o estado global da aplicação, utilizando o React Context API para compartilhar dados entre diferentes componentes de forma eficiente.
- Pages: Representações das páginas do sistema, que agrupam componentes e definem as interfaces principais exibidas aos usuários em diferentes rotas.
- Routes: Configuração das rotas da aplicação, mapeando URLs para páginas específicas, utilizando a biblioteca React Router para navegação interna.
- Services: Arquivos que centralizam a comunicação com as APIs, contendo funções responsáveis por realizar requisições HTTP e manipular os dados recebidos.
- Types: Definições de tipos e interfaces utilizadas na aplicação para garantir maior segurança e consistência durante o desenvolvimento, especialmente ao utilizar TypeScript.

3.6.2 Contextos

A implementação de contextos no projeto foi fundamental para gerenciar estados globais que precisavam ser acessados por múltiplos componentes de forma eficiente e organizada. No projeto, foram desenvolvidos dois contextos principais: o Contexto de Usuário, que gerencia a autenticação, e o Contexto de Formulário de Acomodação, que lida com o preenchimento de dados em múltiplas etapas do cadastro de acomodações.

Contexto de Usuário

O Contexto de Usuário ("UserContext") foi implementado para gerenciar informações relacionadas à autenticação e ao perfil do usuário logado. Este contexto centraliza operações como login, logout, cadastro e verificação de autenticação, além de disponibilizar os dados do usuário para os demais componentes da aplicação que necessitam dessa informação.

Também é responsável pela verificação da validade do *token* em um novo acesso ao sistema, direcionando o usuário para página de login caso necessário.

A utilização do Contexto de Usuário permite que o estado de autenticação seja facilmente acessado por diferentes componentes sem a necessidade de passar *props* manualmente entre eles. Com isso, é possível simplificar a lógica de controle de acesso às páginas e funcionalidades restritas da aplicação.

Contexto de Formulário de Acomodação

O Contexto de Formulário de Acomodação ("*LodgeFormContext*") foi desenvolvido para gerenciar o estado do formulário de acomodação, que é composto por múltiplas etapas. Este contexto facilita o armazenamento e a atualização dos dados inseridos pelo usuário em cada etapa do formulário, garantindo que as informações sejam persistidas de maneira contínua durante a navegação entre as etapas.

O contexto também centraliza a lógica necessária para validar os dados em cada etapa e controlar a navegação entre elas. Isso permite que a lógica do formulário permaneça isolada e reutilizável, promovendo maior organização e manutenção do código.

3.6.3 Roteamento de Páginas

O roteamento de páginas foi implementado utilizando a biblioteca *react-router-dom*, que permite a navegação entre diferentes seções da aplicação de forma dinâmica. O código apresentado a seguir Figura 3.16 mostra a configuração do componente *RootRouter*, responsável por definir as rotas principais do sistema e gerenciar o acesso a páginas públicas e protegidas.

```jsx
export const RootRouter = () => {
  return (
    <BrowserRouter>
      <Routes>
        <Route element={<Layout />}>
          <Route path='/' element={<Home />} />
          <Route path='/login' element={<Login />} />
          <Route path='/sign-up' element={<SignUp />} />
          <Route path='/lodges' element={<Lodges />} />
          <Route path='/lodge/:id' element={<LodgePage />} />
          <Route element={<ProtectedRoute />}>
            {/* Rotas protegidas */}
            <Route path='/profile' element={<Profile />} />
            <Route path='/profile/edit' element={<EditProfile />} />
            <Route path='/my-lodges' element={<UserLodges />} />
            {["/my-lodges/create", "/my-lodges/edit"].map((path, i) => (
              <Route
                key={i}
                path={path}
                element={<LodgeFormProvider children={<LodgeForm />} />}
              />
            ))}
          </Route>
          <Route path='*' element={<NotFound />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
};
```

Figura 3.16: Código de gerenciamento de rotas no cliente usando react-router-dom.

O componente utiliza o *BrowserRouter* para envolver toda a aplicação e o componente *Routes* para definir as rotas. Dentro do componente *Routes*, são especificadas as diferentes páginas disponíveis na aplicação, associando cada rota a um componente correspondente.

Rotas Públicas

A estrutura das rotas está organizada da seguinte forma:

- Rota "/": Página inicial (*Home*).
- Rota "/login": Página de login (*Login*).
- Rota "/sign-up": Página de cadastro de usuário (*SignUp*).

- Rota "/lodges": Página de busca e lista de acomodações (*Lodges*).
- Rota "/lodge/:id": Página de detalhes de uma acomodação específica (*LodgePage*).

Rotas Protegidas

As rotas protegidas são aquelas que requerem autenticação para serem acessadas. Para implementar esse controle, foi utilizado o componente *ProtectedRoute*, que verifica se o usuário está autenticado antes de permitir o acesso às rotas internas.

As principais rotas protegidas são:

- Rota "/profile": Página de perfil do usuário (*Profile*).
- Rota "/profile/edit": Página de edição do perfil do usuário (*EditProfile*).
- Rota "/my-lodges": Página que exibe as acomodações cadastradas pelo usuário (*UserLodges*).
- Rotas "/my-lodges/create" e "/my-lodges/edit": Páginas para criar ou editar acomodações, utilizando o contexto *LodgeFormProvider* para gerenciar o estado do formulário (*LodgeForm*).

Rota de Tratamento de Erros

A aplicação também possui uma rota de *fallback*, que utiliza o componente *NotFound* para exibir uma mensagem quando o usuário tenta acessar uma página inexistente. Essa rota é definida pelo caminho "*".

Benefícios do Roteamento Implementado

A implementação do roteamento trouxe diversos benefícios para a organização e navegação da aplicação:

1. **Modularidade**: As páginas e suas respectivas rotas estão organizadas de forma clara e reutilizável.
2. **Controle de Acesso**: O uso de rotas protegidas garante que apenas usuários autenticados possam acessar determinadas páginas.
3. **Melhor Experiência do Usuário**: A navegação entre páginas é fluida e dinâmica, proporcionando uma melhor experiência para o usuário final. Observando o componente Layout, que engloba todas as páginas, com um cabeçalho e um rodapé.

Com essa configuração, o sistema está preparado para lidar com diferentes fluxos de navegação e gerenciar o acesso a páginas de acordo com o perfil e a autenticidade do usuário.

3.6.4 Responsividade e Compatibilidade

O sistema foi desenvolvido com foco na responsividade e na compatibilidade com os navegadores modernos, utilizando a metodologia Mobile First 2.3.4 e ferramentas de desenvolvimento para este fim. A responsividade foi alcançada por meio do design fluido e das classes de layout do *Bootstrap*, que permitem que a interface se ajuste automaticamente a diferentes tamanhos de tela, proporcionando uma experiência de usuário otimizada em dispositivos móveis, *tablets* e *desktops*.

Além disso, o sistema é totalmente compatível com os principais navegadores modernos, como *Google Chrome, Mozilla Firefox, Safari e Microsoft Edge*, que suportam as tecnologias mais recentes de *JavaScript*, garantindo uma navegação fluida, com atualizações rápidas e sem a necessidade de recarregar a página, melhorando a performance e a experiência geral.

Com isso, o sistema oferece uma interface responsiva, acessível e funcional em uma ampla variedade de dispositivos e navegadores, assegurando que o usuário tenha uma experiência consistente e de alta qualidade em qualquer plataforma.

Na Figura 3.17 é demonstrado como as telas da aplicação foram implementadas para se comportar corretamente em dispositivos com tela pequena, como em celulares. Neste exemplo a tela inicial que se adapta ao tamanho da tela do usuário.

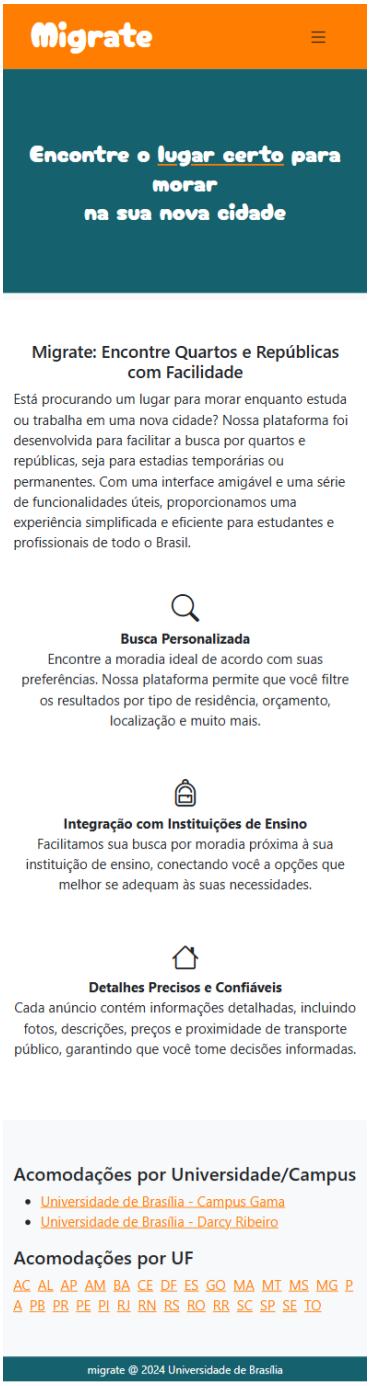


Figura 3.17: Tela inicial da aplicação em dispositivo com tela pequena.

Capítulo 4

A Aplicação Migrate

Neste capítulo, é apresentada a aplicação desenvolvida, detalhando funcionalidades e a forma como cada componente se comporta para atender aos requisitos propostos inicialmente. A aplicação foi desenvolvida para oferecer uma experiência intuitiva, fácil e segura tanto para anunciantes quanto para quem procura acomodação e os resultados obtidos demonstram tanto a viabilidade técnica da solução, quanto sua adequação às necessidades reais dos usuários, cumprindo assim o propósito central do projeto.

Para isso, são apresentados os resultados obtidos a partir da implementação da aplicação, com foco na validação dos requisitos definidos durante a fase de análise de requisitos 3.1. O objetivo é demonstrar por meio das funcionalidades como o sistema desenvolvido atende às necessidades dos usuários, tanto estudantes quanto proprietários de acomodações. Foram organizados os seguintes conjuntos de funcionalidades:

- **Autenticação**: Garante a segurança do sistema, permitindo: criação de contas, login e edição de perfis de usuários.
- **Cadastro de anúncios**: Possibilita a publicação de anúncios de acomodações, com formulários para inserção de detalhes, localização e fotos.
- **Busca de acomodações**: Exibe a lista de anúncios disponíveis, com recursos de filtragem, paginação e organização dos resultados.
- **Visualização de anúncio**: Apresenta informações detalhadas da acomodação, incluindo fotos, descrição, localização, dados de contato e um mapa interativo.
- **Edição de anúncio**: Permite que usuários atualizem ou removam seus anúncios por meio de formulários editáveis.

A seguir, estas funcionalidades são descritas destacando o cumprimento dos requisitos funcionais e não funcionais, além de aspectos relacionados à usabilidade e desempenho

da aplicação, evidenciando como cada requisito foi transformado em uma funcionalidade operacional.

4.1 Autenticação

O primeiro requisito funcional a ser testado é o cadastro de usuário (**RF01**). Para isso, foram utilizados dados de um usuário fictício. Inicialmente, foi simulada uma tentativa de cadastro com informações inválidas, com campos obrigatórios em branco. Como mostra a Figura 4.1, o sistema exibe mensagens de erro, indicando os problemas nos dados inseridos. Em seguida, foi preenchido o formulário corretamente com dados válidos, conforme ilustrado na Figura 4.2. Por fim, ao submeter o formulário, o sistema redireciona o usuário para uma tela de confirmação, exibindo a mensagem de "Conta criada com sucesso", como evidenciado na Figura 4.3. Esse fluxo completo valida a funcionalidade de cadastro de usuário, e também a experiência do usuário, garantindo que o sistema oriente adequadamente o usuário em caso de erros e confirme o sucesso da operação.

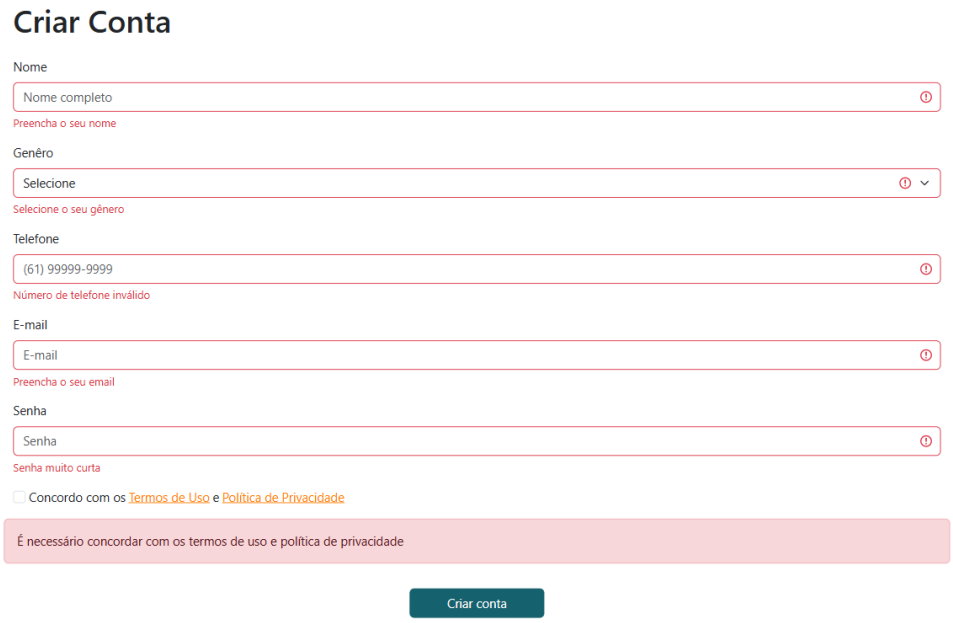


Figura 4.1: Formulário de cadastro apontando erros de preenchimento.

O próximo requisito testado é o login por e-mail e senha (**RF02**). Para validar essa funcionalidade, foi iniciada uma tentativa de login utilizando credenciais incorretas, onde o sistema exibe uma mensagem de erro informando que as credenciais estão incorretas. Em seguida, foram inseridos os dados corretos do usuário cadastrado anteriormente, como mostrado na Figura 4.4, utilizando um e-mail e senha previamente cadastrados. Ao

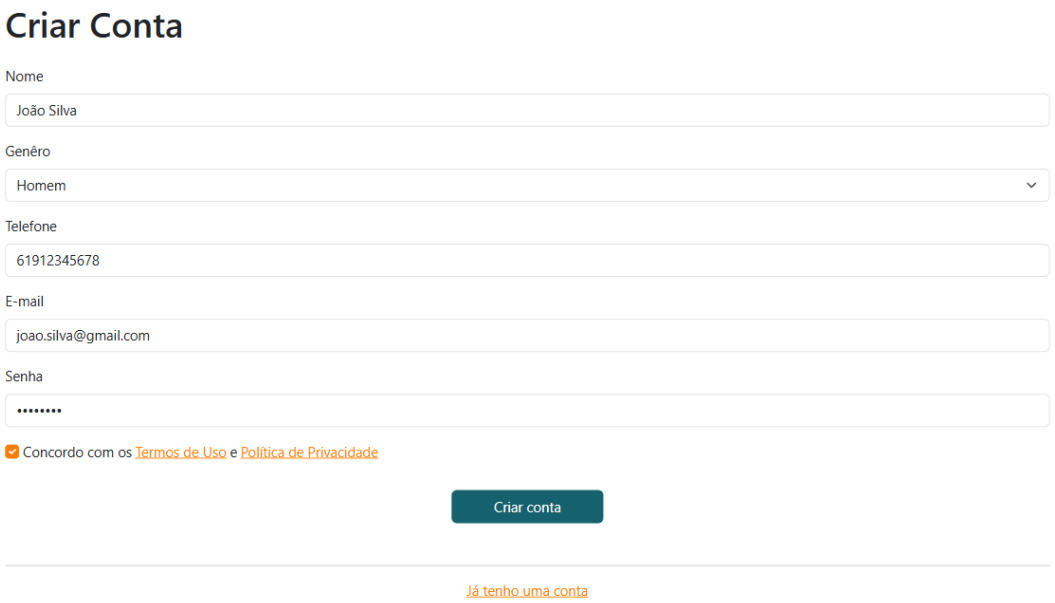


Figura 4.2: Formulário de cadastro com dados corretamente preenchidos.

submeter o formulário, o sistema autentica o usuário e o redireciona para a tela de perfil do usuário, conforme evidenciado na Figura 4.5. Esse fluxo não apenas valida a autenticação segura, mas também garante que o sistema forneça feedback adequado ao usuário em caso de falha e uma transição suave para a área logada após o sucesso.

Na Figura 4.6 pode-se observar a validação do requisito não funcional **RNF02**, que prevê a existência de uma interface para documentar e interagir com o back-end, implementado por meio da integração do Swagger UI ao projeto. Nela também pode-se validar visualmente (ícone de cadeado) que as rotas de login e cadastro são públicas, ou seja, acessível por qualquer usuário. Já rotas de gerenciamento do perfil do usuário é necessária autenticação.

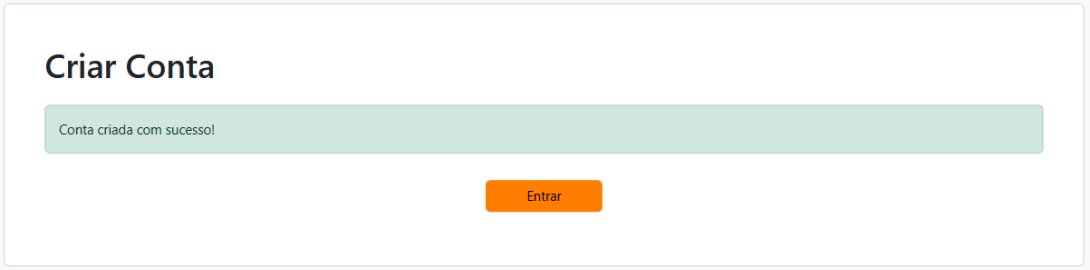


Figura 4.3: Aviso de conta criada com sucesso.



Figura 4.4: Formulário de autenticação preenchido.

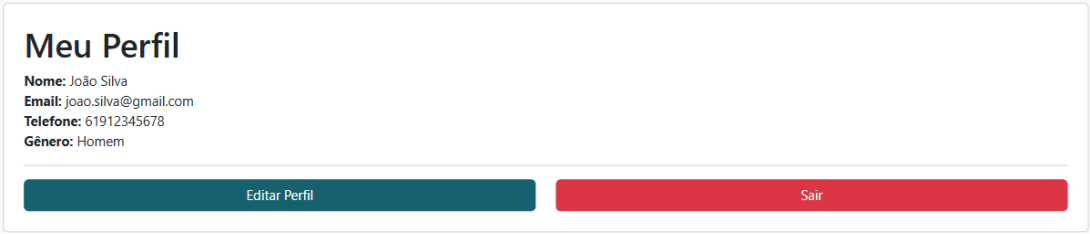


Figura 4.5: Tela de perfil do usuário logado.

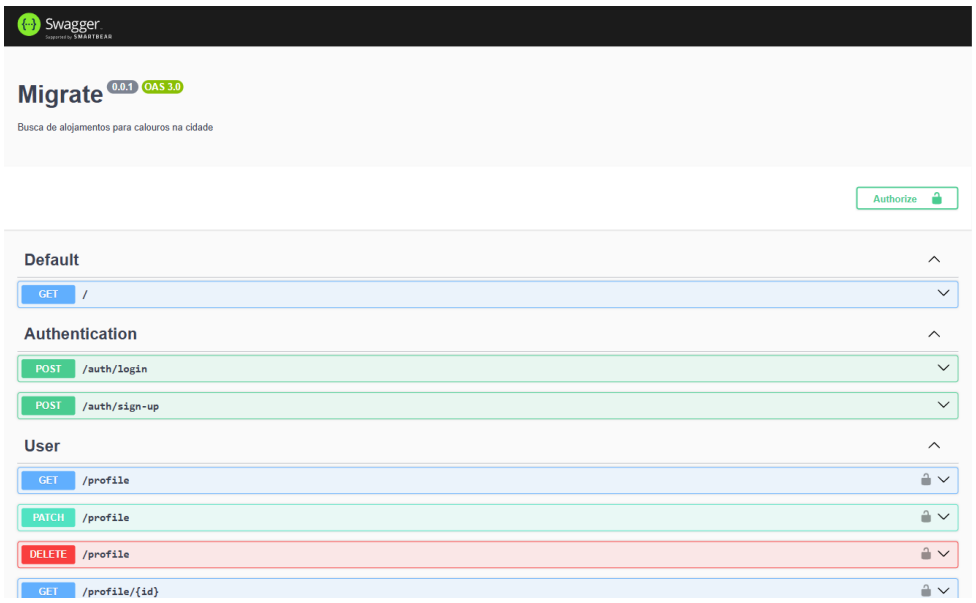


Figura 4.6: Swagger UI da aplicação exibindo rotas de autenticação e de usuário.

4.2 Cadastro de Anúncios

O cadastro de anúncios de acomodações é um requisito do sistema (**RF03**), que só é permitido para usuários autenticados. Na tela "Meus Anúncios" caso o usuário não tenha nenhum anúncio cadastrado é exibida uma mensagem e a opção de criar novo anúncio, como ilustrado na Figura 4.7.

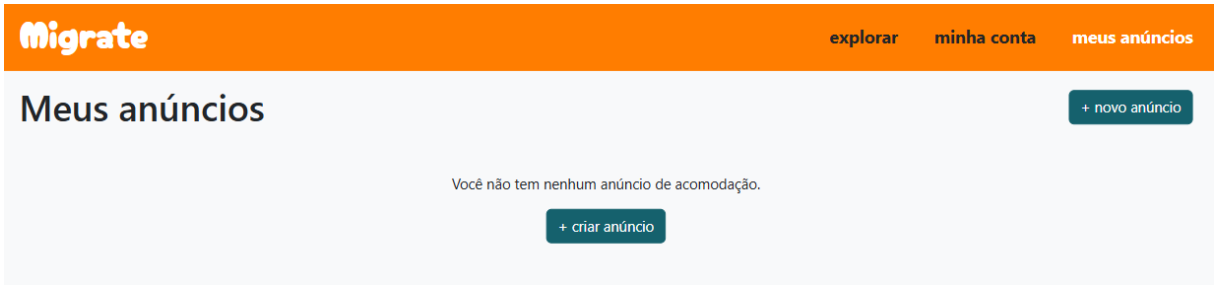


Figura 4.7: Tela de anúncios de acomodações do usuário.

Ao selecionar a opção de criar novo anúncio o usuário é redirecionado para o formulário de criação de anúncio de acomodação, o formulário é dividido em três etapas para diminuir a complexidade no preenchimento de dados para o usuário. A primeira etapa, representada na Figura 4.8 possui campos de informações gerais: título, tipo de espaço, tipo de acomodação e descrição, e também informações adicionais: informação de contato, quais os gêneros aceitos na acomodação, valor mensal e se aquela acomodação tem relação com alguma instituição de ensino. Seguindo a validação do requisito funcional **RF04**.

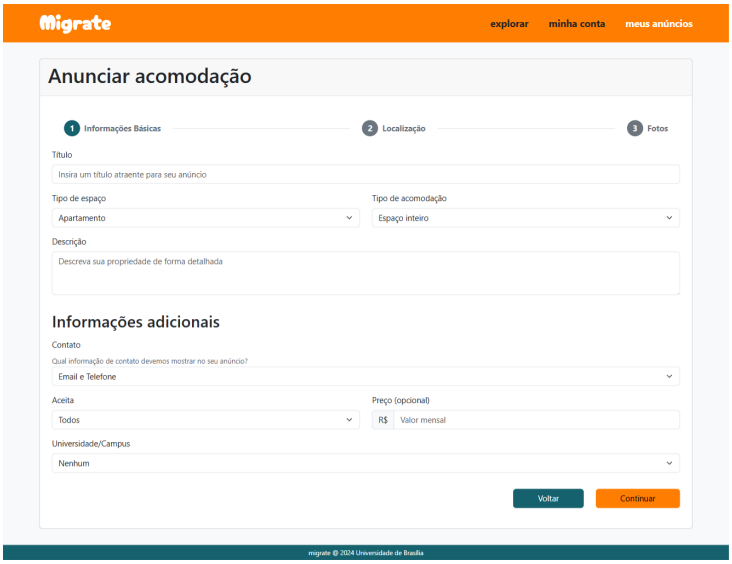


Figura 4.8: Primeira etapa do formulário de cadastro de anúncio de acomodação.

O formulário de cadastro de acomodação é pré-preenchido com valores padrão em alguns campos, os quais o usuário pode modificar conforme necessário. No entanto, o sistema impede o envio do formulário caso haja campos obrigatórios em branco, exibindo mensagens de erro para orientar o usuário. Conforme ilustrado na Figura 4.9, é possível visualizar o preenchimento das informações gerais de uma acomodação fictícia. Neste exemplo, uma instituição de ensino previamente cadastrada no sistema foi selecionada (**RF06**), o que automaticamente exibe o campo de "forma de transporte". Essa informação será utilizada para calcular a rota e o tempo de deslocamento da acomodação até a instituição de ensino, funcionalidade utilizada na exibição do mapa no anúncio.

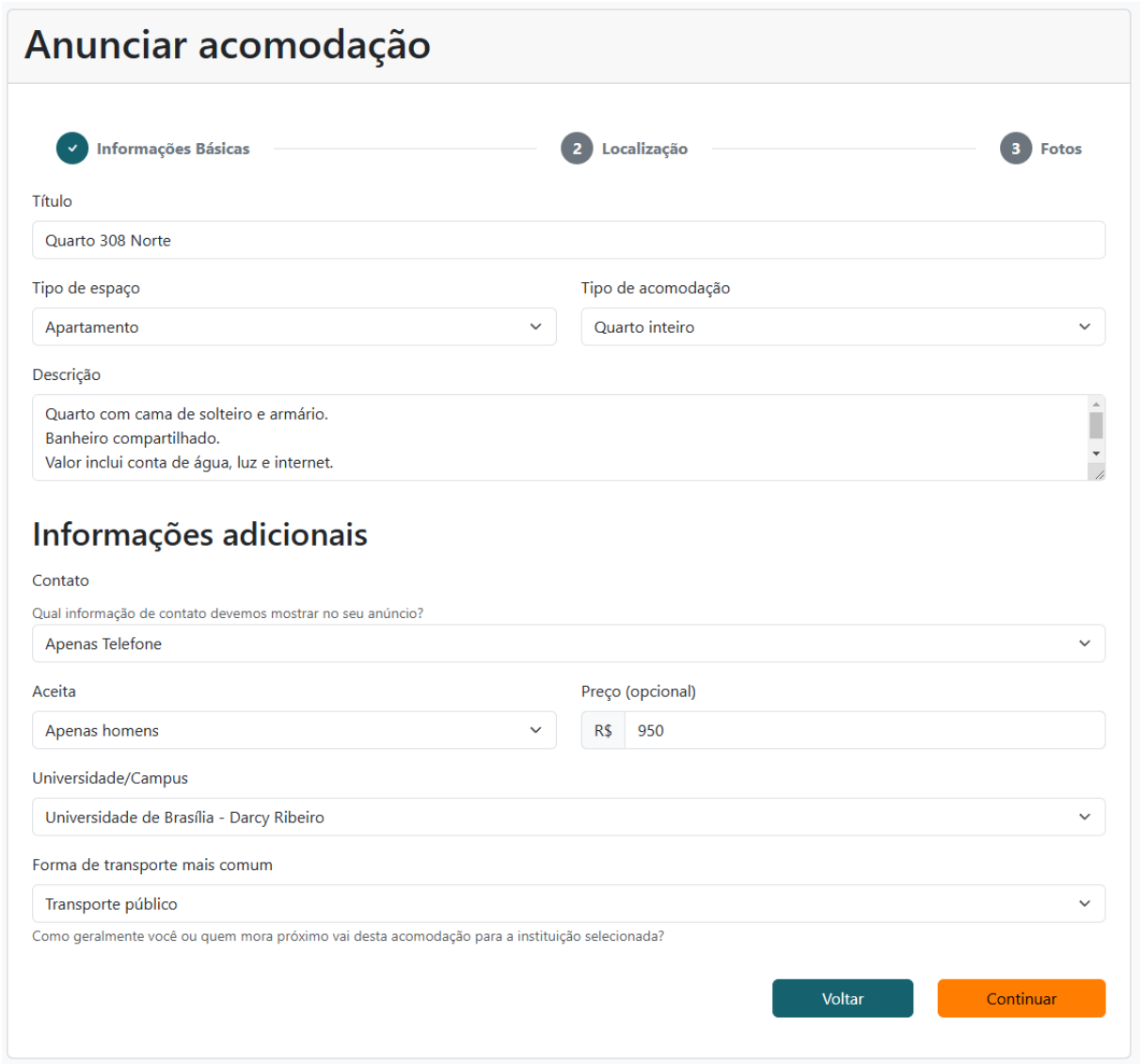


Figura 4.9: Formulário de cadastro de acomodação preenchido.

A próxima etapa do cadastro é a localização da acomodação. Nessa fase, ao preencher

o campo do CEP, o sistema realiza uma requisição a um serviço de consulta de Código de Endereçamento Postal (CEP) do Brasil (ViaCEP). Com base nos dados retornados, os campos de endereço, bairro, cidade e estado, são automaticamente preenchidos, conforme ilustrado na Figura 4.10. Essa funcionalidade agiliza o processo de cadastro e reduz erros de digitação.

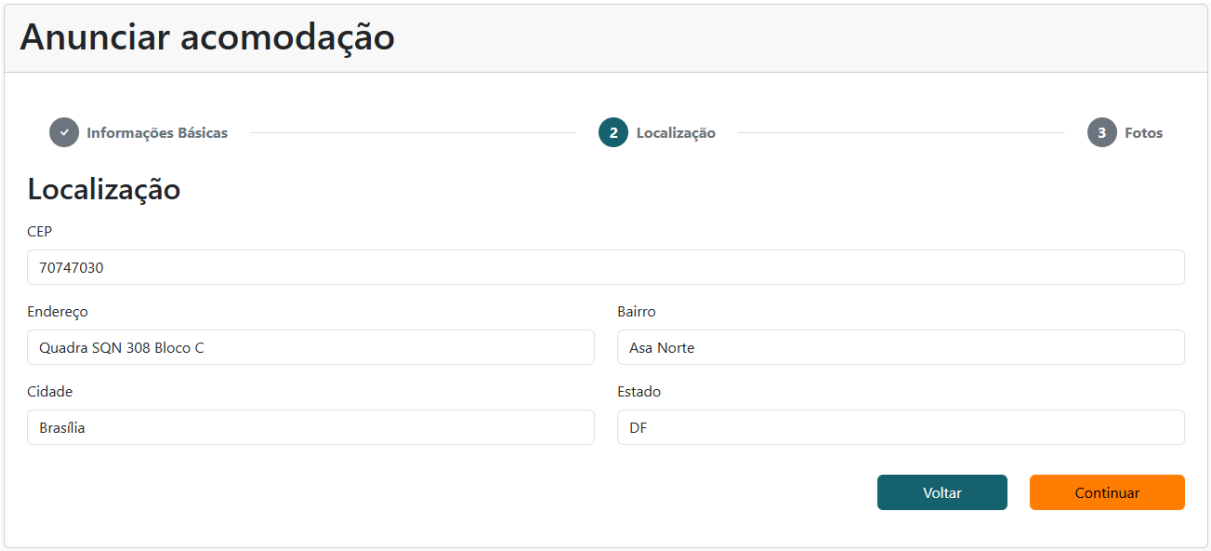


Figura 4.10: Etapa de cadastro da localização da acomodação.

A terceira e última etapa consiste no cadastro das fotos do anúncio. O sistema exige o envio de pelo menos uma foto e permite o upload de até três imagens por anúncio (**RF05**). Além disso, cada foto não pode ultrapassar o tamanho máximo de **200 kB**, conforme estabelecido pelo requisito não funcional **RNF03**. Na Figura 4.11, é possível observar o aviso exibido ao usuário caso ele tente enviar uma imagem com tamanho superior ao permitido. Por outro lado, a Figura 4.12 exemplifica o envio de três imagens válidas, com o cadastro do anúncio pronto para ser finalizado e salvo no banco de dados.

Após o usuário clicar no botão "Criar"a acomodação é salva no banco de dados e as imagens no Minio 2.4.7. Redirecionando o usuário para a página "Meus Anúncios".

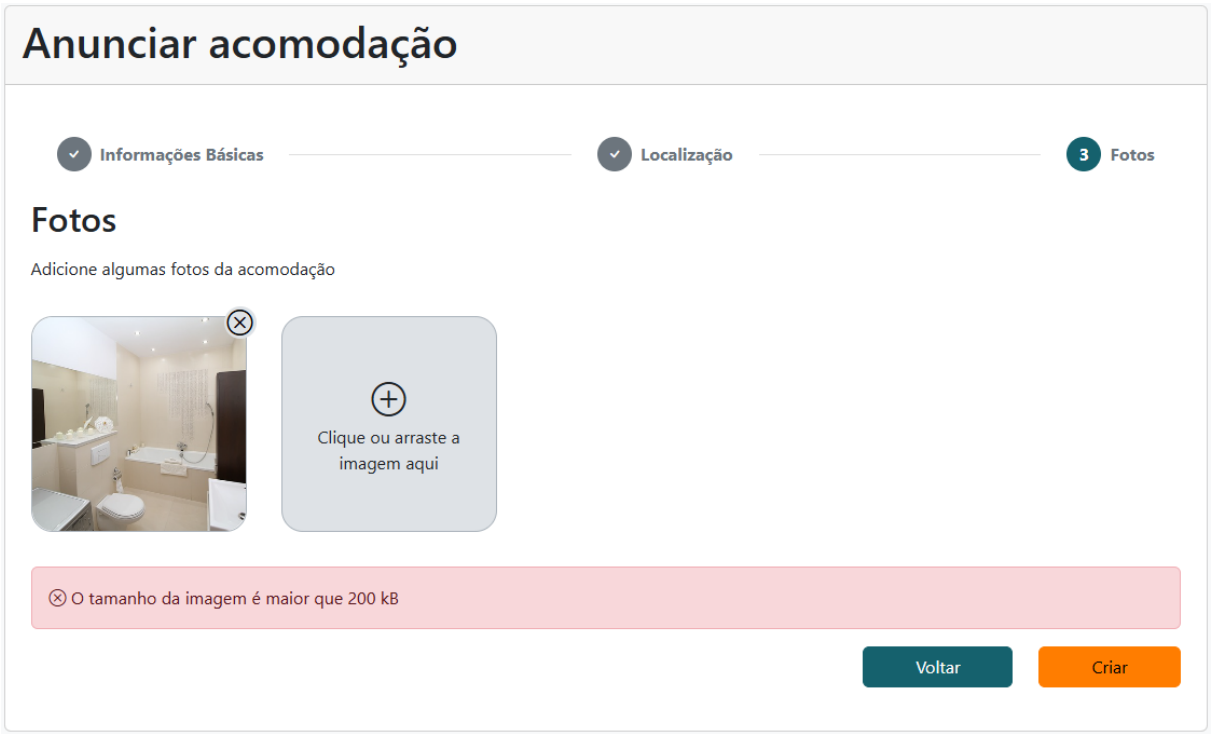


Figura 4.11: Etapa de cadastro de fotos com erro.

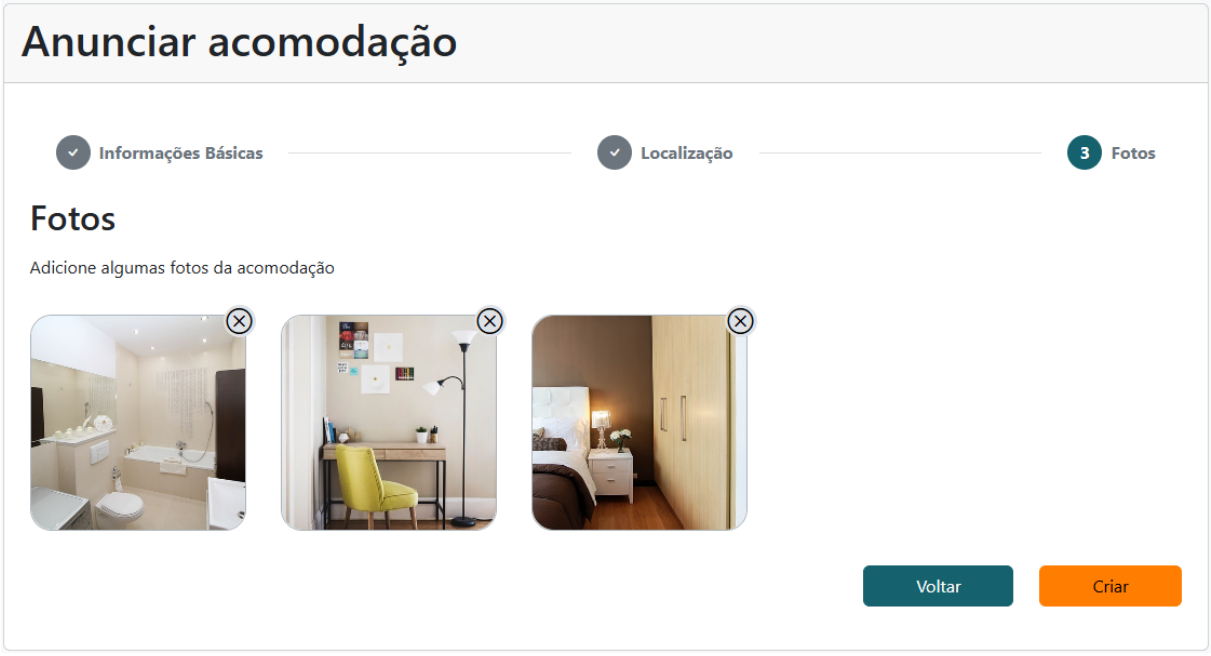


Figura 4.12: Etapa de cadastro de fotos preenchido com 3 imagens.

4.3 Busca de Anúncios

O item do menu superior “Explorar” leva o usuário a página de busca de anúncios, representada na Figura 4.13.

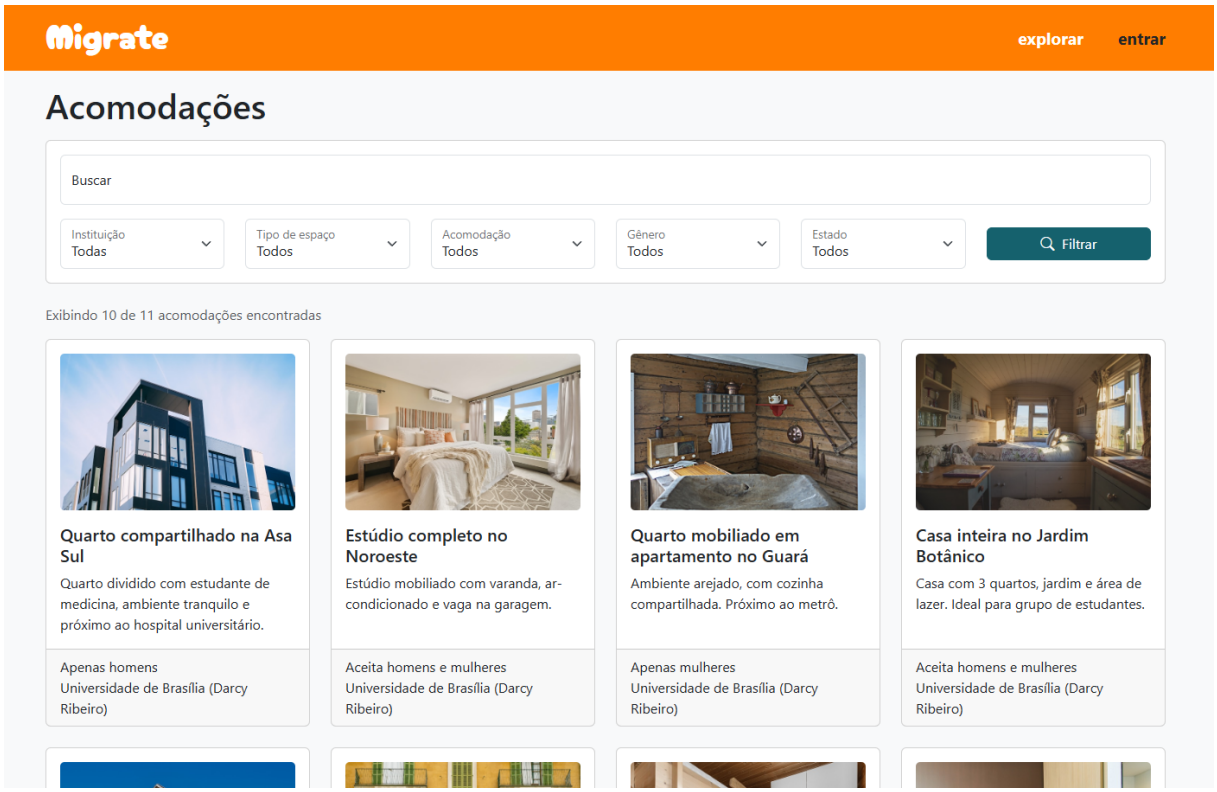


Figura 4.13: Página de explorar acomodações cadastradas.

Nela é possível visualizar a listagem de acomodações, exibidas em ordem de criação e de forma páginada (**RF08**). Cada acomodação é representada por um *card* contendo a imagem, título, descrição e qual a instituição de ensino pertence (**RF09**). Além disso na parte superior da página há uma seção para filtragem de acomodações, sendo possível que os estudantes filtrem os anúncios por estado, gênero, tipo de acomodação, instituição e título (**RF07**).

Cada um dos *cards* de acomodações é clicável, levando o usuário a página de detalhes da acomodação selecionada (**RF10**).

4.4 Visualização de Anúncios

Como visto anteriormente, ao selecionar um anúncio de acomodação na listagem de acomodações o usuário é redirecionada a página de detalhes do anúncio (**RF11**) Figura 4.14.

Neste exemplo (Figura 4.14) é possível visualizar o título, descrição, carrossel de fotos da acomodação, valor mensal, endereço, tipo de acomodação (neste caso um quarto em apartamento) e quais gêneros são aceitos nesta acomodação (neste caso apenas homens).

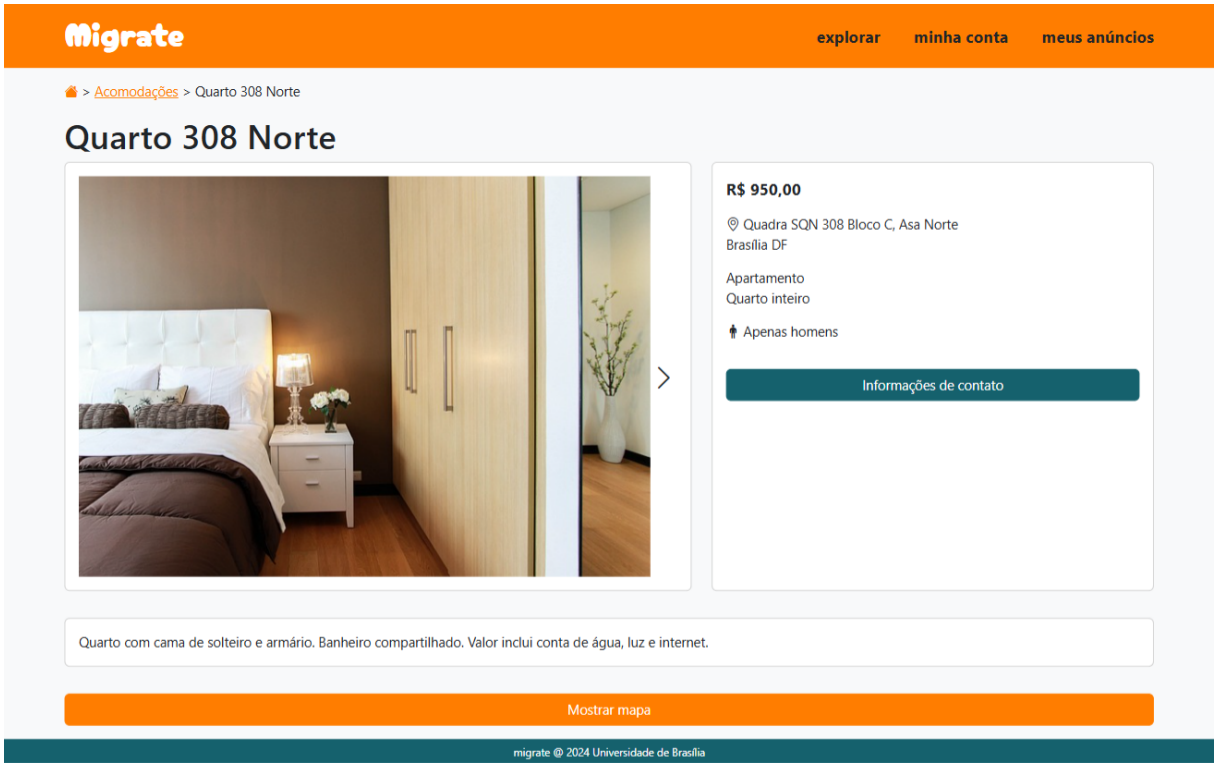


Figura 4.14: Página de detalhes da acomodação.

Além disso botões de “Informações de contato” e de “Mostrar mapa”, sendo estas funcionalidades permitidas apenas para usuários autenticados, a fim de proteger dados sensíveis dos usuários.

Caso um usuário autenticado clique em “Informações de contato” são exibidos os dados de telefone e/ou e-mail do anunciante (**RF14**). Já ao clicar em “Mostrar mapa” é carregado um mapa interativo (**RF12**), utilizando a API do Google, com a localização exata da acomodação. E caso possua uma instituição de ensino associada são exibidas as rotas de transporte da acomodação até a instituição (**RF13**), com tempo estimado e distância a ser percorrida Figura 4.15.

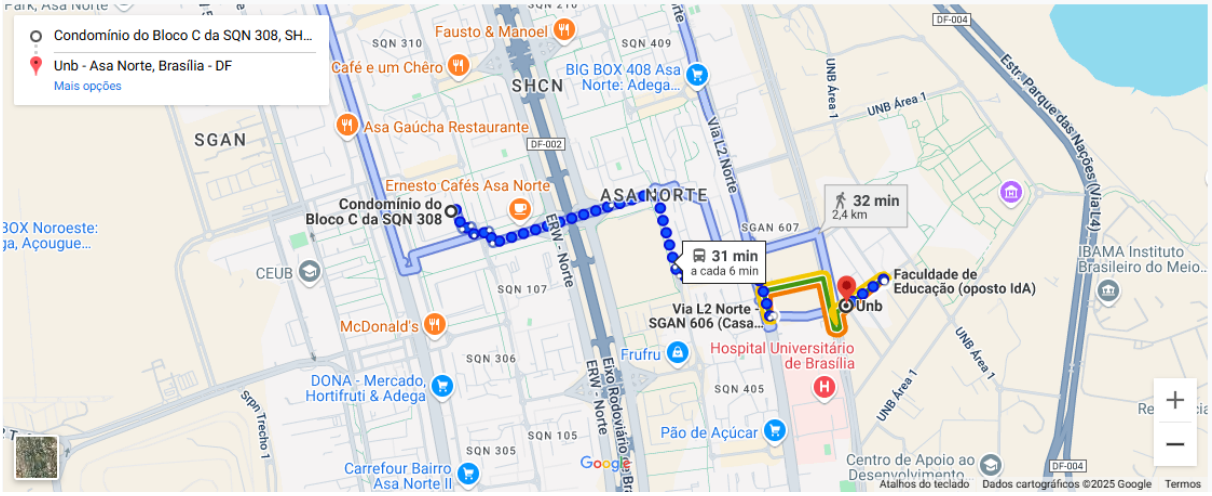


Figura 4.15: Mapa com rotas da acomodação à instituição.

4.5 Edição de Anúncio

Os próximos requisitos a serem validados estão relacionados à edição de um anúncio já criado. Conforme detalhado na subseção 4.2, um anúncio de acomodação com dados fictícios foi criado com sucesso. Nesta etapa, serão validadas as funcionalidades de alteração de dados (**RF15**), remoção e adição de fotos (**RF16**) e exclusão do anúncio (**RF17**), garantindo que o registro permaneça no banco de dados, conforme estabelecido pelo requisito não funcional **RNF04**. Na Figura 4.16, é possível visualizar a lista de anúncios cadastrados pelo usuário, que exibe o título, a descrição e as imagens de cada acomodação. Além disso, são apresentados três botões de ação para cada anúncio: um ícone de lixeira vermelha para exclusão, um ícone de lápis laranja para edição e um ícone de olho azul para visualização detalhada do anúncio.

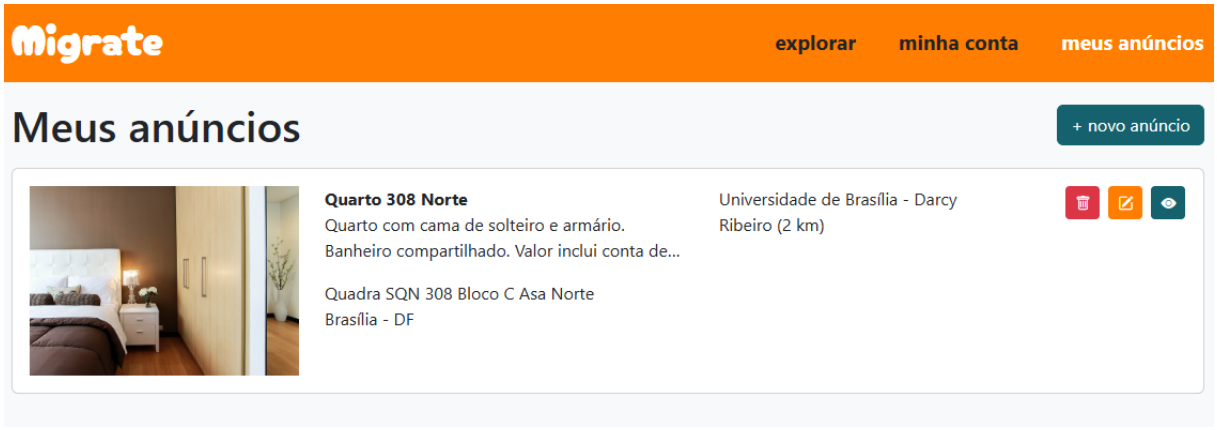


Figura 4.16: Tela “Meus Anúncios” com um anúncio cadastrado.

4.6 Demonstração Prática

Para complementar a explicação teórica e oferecer uma visão prática do sistema em funcionamento, foi preparado um vídeo de demonstração que ilustra, de forma dinâmica, todas as funcionalidades implementadas. Neste material, é possível acompanhar o fluxo completo da aplicação — desde o cadastro e autenticação de usuários até a criação, edição e busca de anúncios —, destacando a interface intuitiva, a interatividade dos componentes e a integração entre as diferentes funcionalidades. O vídeo está disponível no seguinte link:

`https://drive.google.com/file/d/1-fQnOZ5isSblVOTHGvWvD5kTHM9KoO98/view?usp=drive_link`

A visualização é recomendada para uma compreensão mais clara da experiência do usuário e da eficácia das soluções adotadas no desenvolvimento.

No capítulo seguinte, serão apresentadas as conclusões do trabalho, sintetizando os objetivos alcançados e as contribuições do projeto. Além disso, serão discutidas possíveis direções futuras para expansão e aprimoramento do sistema, com base nas lições aprendidas e nas demandas identificadas ao longo da pesquisa.

Capítulo 5

Conclusões

Este trabalho teve como objetivo geral desenvolver uma solução tecnológica para facilitar a adaptação de calouros universitários em novas cidades, centralizando informações sobre moradias e mobilidade para instituições de ensino, buscando resolver um problema real enfrentado por estes estudantes.

Para isso, foram utilizadas ferramentas modernas de desenvolvimento: o MySQL como banco de dados relacional, garantindo integridade dos dados e consultas eficientes; o NestJS para a construção do servidor, proporcionando uma arquitetura modular e de fácil manutenção; o React permitindo a criação de uma interface de usuário dinâmica e responsiva; e o Docker para conteinerização, assegurando portabilidade entre ambientes e escalabilidade dos serviços.

Desde a percepção do problema até a implementação da solução, foi possível compreender as diversas etapas, metodologias e desafios envolvidos no processo. O resultado foi um aplicativo que cumpriu com os objetivos iniciais, oferecendo uma plataforma centralizada e eficiente para auxiliar na adaptação dos estudantes. Durante o desenvolvimento, priorizaram-se questões críticas identificadas, como facilidade de uso, localidade e mobilidade, que direcionaram as funcionalidades que entregam valor ao usuário e a usabilidade do sistema.

A implementação focou em uma interface intuitiva e acessível, projetada para diferentes níveis de familiaridade tecnológica, permitindo navegação simplificada. Além disso, integrou-se medidas de segurança para proteção dos dados dos usuários, mitigando riscos de vazamentos e ações não autorizadas.

O aplicativo desenvolvido procurou não apenas simplificar a busca por moradias estudantis, mas também promover segurança, conforto e inclusão, contribuindo para uma transição mais tranquila e bem sucedida dos estudantes em seu novo ambiente acadêmico.

5.1 Trabalhos Futuros

Para trabalhos futuros é sugerida a implementação de novas funcionalidades para aumentar a usabilidade, segurança e variações de uso. Pode-se melhorar o sistema de autenticação o deixando mais seguro e confiável, como por exemplo adicionar funcionalidades de confirmação de e-mail ou número de telefone, recuperação de senha e *CAPTCHA* para restringir usuários não humanos.

Também é possível implementar funcionalidades que permitam usuários que estão a procura de repúblicas se mostrem disponíveis no sistema, abrindo a possibilidade de que estudantes que possuam interesses semelhantes e estão sem uma acomodação possam se agrupar para criar uma nova república.

Outra sugestão para aumentar tanto a colaboração quanto a segurança do sistema é permitir que usuários façam comentários, avaliações e denúncias nos anúncios de acomodações.

Referências

[1] Cardoso, Renata, Andressa Castro, Gustavo Frio e Adelar Fochezatto: *Migração estudantil: uma análise do impacto da política de cotas e do programa universidade para todos*. Em *Universidade e território: ensino superior e desenvolvimento regional no Brasil do século XXI*, páginas 429–456. Instituto de Pesquisa Econômica Aplicada (Ipea), 2022. 1, 5, 6

[2] Lacerda, Izabella e Felipe Valentini: *Impacto da moradia estudantil no desempenho acadêmico e na permanência na universidade*. Psicologia Escolar e Educacional, 22(2):413–423, 2018. 1

[3] Szerman, Christiane e Cecília M. Berriel: *The effects of a centralized college admission mechanism on migration and college enrollment: Evidence from brazil*. Em *37th Meeting of the Brazilian Econometric Society*, Florianópolis, 2015. SBE Meetings. 2, 3

[4] Womack, James P. e Daniel T. Jones: *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. Simon Schuster, 1996. 8

[5] Beck, Kent.: *Test-Driven Development: By Example*. Addison-Wesley, 2003. 8

[6] Tanenbaum, Andrew S.: *Redes de Computadores 5a ed.* Pearson, 2011. 9

[7] Pressman, Roger S.: *Engenharia de Software: uma abordagem profissional 8a ed.* AMGH, 2016. 10

[8] Microsoft: *Typescript*. `https://www.typescriptlang.org/docs/`, acesso em 2025-01-04. 11

[9] React: *React - a javascript library for building user interfaces.* `https://react.dev`, acesso em 2025-01-04. 11, 12

[10] Bootstrap: *Get started with bootstrap*. `https://getbootstrap.com/docs/5.3/getting-started/introduction/`. 13

[11] NestJs: *Documentation*. `https://docs.nestjs.com/`. 13, 31

[12] Prat, Nicolas: *Best practices for api documentation with swagger and openapi*. Journal of Software Engineering, 15:120–135, 2020. 14

[13] Jest: *Delightful javascript testing*. `https://jestjs.io/`. 14

[14] Documentation, MinIO: *Minio object storage documentation*. https://min.io/docs, acesso em 2025-01-05. 14

[15] Oracle: *O que é o mysql*. https://www.oracle.com/br/mysql/what-is-mysql/. 15

[16] Google: *Google maps platform documentation*. https://developers.google.com/maps/documentation, acesso em 2025-01-04. 15

[17] Docker: *What is docker?* https://docs.docker.com/get-started/docker-overview/, acesso em 2025-01-06. 15

[18] DNC, Escola: *Figma: O que é e como usar?* https://www.escoladnc.com.br/blog/figma-o-que-e-e-como-usar. 15, 26

[19] Auth0: *Introduction to json web tokens*. https://jwt.io/introduction, acesso em 2025-01-06. 16

[20] Bradbury, D.: *Token-based authentication using jwt*. Manning Publications, 2016. 17

[21] Auth0: *Adding salt to hashing: A better way to store passwords*. https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords. 17, 32

[22] Foundation, OWASP: *Owasp top ten web application security risks*. https://owasp.org/www-project-top-ten. 17

[23] Brasil: *Lei geral de proteção de dados pessoais, lei nº 13.709, de 14 de agosto de 2018*. 18, 19

[24] GDPR.EU: *General data protection regulation*. https://gdpr.eu/, acesso em 2025-01-05. 18, 20

[25] Serpro: *Guia prático da lgpd*. https://www.serpro.gov.br/lgpd, acesso em 2025-01-05. 18, 19

[26] Facebook: *Create react app*. https://github.com/facebook/create-react-app. 31

Apêndice A

Política de Privacidade

Política de Privacidade

Esta Política de Privacidade descreve como o nosso software (*Migrate*) coleta, utiliza, armazena e protege os dados pessoais dos usuários. Nosso compromisso é garantir a privacidade e a segurança das informações fornecidas por nossos usuários, em conformidade com a Lei Geral de Proteção de Dados Pessoais (LGPD).

1. Dados Coletados

Migrate coleta as seguintes informações pessoais dos usuários:

- Nome completo
- Endereço de e-mail
- Número de telefone
- Endereço da moradia

Esses dados são fornecidos pelos próprios usuários durante os processos de cadastro na plataforma.

2. Finalidade do Tratamento dos Dados

Os dados coletados são utilizados para as seguintes finalidades:

- Identificação e autenticação do usuário no sistema;
- Melhoria da experiência do usuário na plataforma;
- Informações de contato para interessados na moradia.

3. Armazenamento dos Dados

Os dados pessoais dos usuários são armazenados de forma segura em nosso banco de dados, utilizando medidas técnicas adequadas para garantir sua proteção contra acessos não autorizados, perdas, alterações ou vazamentos.

4. Compartilhamento de Dados

O software **não** compartilha, vende ou aluga os dados pessoais dos usuários a terceiros. Os dados serão utilizados apenas para os fins descritos nesta política.

5. Direitos dos Usuários

Em conformidade com a LGPD, os usuários têm os seguintes direitos em relação aos seus dados pessoais:

- Acesso aos dados pessoais armazenados;
- Correção de dados incompletos, inexatos ou desatualizados;
- Solicitação de exclusão dos dados pessoais, salvo quando a manutenção for exigida por lei;
- Portabilidade dos dados a outro fornecedor de serviço, mediante solicitação.

6. Segurança dos Dados

Implementamos medidas técnicas apropriadas para proteger os dados pessoais dos usuários contra acessos não autorizados, uso indevido, perda ou destruição. Utilizamos criptografia, autenticação segura e outros mecanismos de segurança para garantir a integridade dos dados.

7. Alterações na Política de Privacidade

Esta Política de Privacidade pode ser atualizada periodicamente para refletir mudanças na legislação ou em nossos processos internos. Os usuários serão notificados sobre alterações significativas por meio de avisos no software.

8. Contato

Em caso de dúvidas ou solicitações relacionadas a esta Política de Privacidade, entre em contato conosco pelo e-mail: **180122231@aluno.unb.br**