UNIVERSIDADE FEDERAL DE SANTA CATARINA

VAGOU AQUI: DESENVOLVIMENTO DE UM SISTEMA WEB DE AGREGAÇÃO DE VAGAS DE MORADIA VOLTADO PARA A COMUNIDADE ACADÊMICA DA UFSC

José Ribamar Marçal Martins Júnior

Florianópolis/SC
2023/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Vagou Aqui - Desenvolvimento de um sistema web de agregação de vagas de moradia voltado para a comunidade acadêmica da UFSC

José Ribamar Marçal Martins Júnior

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Florianópolis/SC
2023/2

José Ribamar Marçal Martins Júnior

Vagou Aqui' - Desenvolvimento de um sistema web de agregação de vagas de moradia voltado para a comunidade acadêmica da UFSC

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em SISTEMAS DE INFORMAÇÃO

Orientador(a):

_____

Prof. CRISTIAN KOLIVER

Banca examinadora:

_____

Prof. ISMAEL SEIDEL

_____

Prof. ANDRÉ WUST ZIBETTI

DEDICATÓRIA

Dedico este trabalho ao meu esposo, Vagner, por ser a força constante que impulsiona meus sonhos. Sua inabalável confiança em mim, apoio incansável e torcida pela minha vitória são a luz que guia meus passos. Juntos, superamos desafios, celebramos conquistas e construímos uma jornada marcada por amor e cumplicidade.

Aos meus pais Olga e Marçal, que me deram o presente da vida, bem como uma base sólida de educação e apoio e também aos meus familiares, em especial minha irmã Amanda e minha tia Bia, cujas lições de vida moldaram meu caráter e me ensinaram a perseverar diante das adversidades.

Ao meu amigo Luíz, cuja amizade é um tesouro inestimável. Agradeço por seu apoio nos trabalhos em equipe, pelos valiosos conselhos tanto profissionais quanto pessoais, e por compartilhar comigo as alegrias e desafios da jornada acadêmica.

Ao meu orientador, Cristian Koliver, cuja orientação e apoio foram cruciais para o desenvolvimento deste projeto ao longo do ano. Sua dedicação, expertise e incentivo foram pilares fundamentais para o sucesso desta empreitada acadêmica.

Este trabalho é, em grande parte, uma expressão de gratidão a todos aqueles que, de diversas formas, contribuíram para a sua concretização.

A todos, o meu mais sincero agradecimento.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE REDUÇÕES (abreviaturas, siglas e símbolos)

API - Application programming Interface
AJAX - Asynchronous Javascript and XML
AWS - Amazon Web Services
CLI - interface de linha de comando
CSS - Cascading Style Sheets
CSV - Comma Separated Values
CAGR - Controle Acadêmico da Graduação
DOM - Document Object Model
EC2 - Elastic Compute Cloud
HTML - HyperText Markup Language
HTTP - HyperText Transfer Protocol
HTTPS - HyperText Transfer Protocol Secure
IaaS - Infrastructure as a Service
IP - Internet Protocol
URL - Uniform Resource Locator
JS - Javascript
JSON - JavaScript Object Notation
MVC - Model-View-Controller
PaaS - Plataform as a Service
PHP - PHP: Hypertext Preprocessor
RDS - Relational Database Service
S3 - Simple Storage Service
SaaS - Software as a Service
SFC - Single File Components
TCP - Transmission Control Protocol
TCC - Trabalho de Conclusão de Curso
UI - User Interface
UCS - Universal Coded Character Set
UFSC - Universidade Federal de Santa Catarina
UTF-8 - UCS Transformation Format 8
XPath - XML Path Language
XML - eXtensible Markup Language
YAML - YAML Ain't Markup Language
RS - Recommendation System
LCRM - Latent Class Regression Model
IR - Information Recovery
ROC - Receiver Operating Characteristic
ESA - Análise Semântica Explícita
IGP-M - Índice Geral de Preços – Mercado
MVP - Minimum viable product
JWT - Json Web Token

RESUMO

O presente trabalho documenta todas as etapas de desenvolvimento da aplicação web *"VagouAqui"*, a qual visa ser uma solução abrangente para a desafiadora busca por moradia para aluguel em Florianópolis. Os anúncios são obtidos através de *scrapers* que realizam a coleta de dados de cinco fontes distintas. Os usuários podem se cadastrar na aplicação e podem usar a funcionalidade da aplicação na qual anúncios baseados nas preferências de usuário são exibidos.

A aplicação foi desenvolvida com a arquitetura cliente-servidor, além de utilizar o padrão de programação MVC. O *front-end*, baseado em React.js, oferece uma interface amigável para os usuários. Já o *back-end* foi implementado em *Node.js* e desempenha um papel crucial na integração de dados provenientes dos *scrapers*, simplificando assim a obtenção de informações de várias fontes. A aplicação também oferece uma função de recomendação que proporciona sugestões personalizadas com base nas preferências individuais dos usuários. É importante ressaltar que, embora a intenção inicial fosse implantar a aplicação na nuvem AWS, esse objetivo específico não foi alcançado durante o desenvolvimento. A possibilidade de integração na AWS permanece como uma sugestão valiosa para trabalhos futuros.

Palavras-chave: Sistemas Web, Web Scraping, Javascript, AWS, jQuery, cheerio, axios, jwt, HTML, CSS, Moradia Compartilhada, Estudantil, República Estudantil.

ABSTRACT

This work documents all stages of development of the "VagouAqui" web application, which aims to be a comprehensive solution for the challenging search for rental housing in Florianópolis. The ads are obtained through scrapers that scrape data from five different sources. Users can register on the application and can use the functionality of the application in which advertisements based on user preferences are displayed.

The application was developed with client-server architecture, in addition to using the MVC programming pattern. The front-end, based on React.js, offers a user-friendly interface. The back-end was implemented in Node.js and plays a crucial role in integrating data from scrapers, thus simplifying obtaining information from various sources. The application also offers a recommendation function that provides personalized suggestions based on individual user preferences. It is important to highlight that, although the initial intention was to deploy the application in the AWS cloud, this specific objective was not achieved during development. The possibility of integration into AWS remains a valuable suggestion for future work.

Keywords: Web Systems, Web Scraping, Javascript, AWS, jQuery, cheerio, axios, jwt, HTML, CSS, Shared Housing, Student, Student Republic.

1 INTRODUÇÃO

Nesta seção será apresentado o panorama atual sobre a moradia para os estudantes da UFSC, bem como a justificativa, objetivos e a metodologia empregada no trabalho.

1.1 APRESENTAÇÃO E JUSTIFICATIVA

A UFSC, classificada como a sexta melhor universidade do Brasil (FORBES, 2022), atrai milhares de estudantes para Florianópolis, uma cidade ranqueada como a quinta melhor do país (Austin Rating, ISTO É, 2022). Essa demanda por moradia na ilha de Santa Catarina e nas cidades próximas é impulsionada pela busca de aproximadamente 22,4% dos aprovados no vestibular UFSC de 2021 por alojamento independente (ANDIFES, 2018).

No entanto, a escassez de áreas disponíveis e o aumento das ocupações informais tornaram a busca por moradia um desafio, especialmente para estudantes universitários. O acesso à habitação muitas vezes é informal, com a divisão do aluguel entre colegas, dada a discrepância entre os preços dos aluguéis e a realidade estudantil.

Encontrar um imóvel acessível e bem localizado próximo à universidade é o principal desafio para os novos universitários da UFSC. A busca se torna árdua devido à dispersão dos anúncios de aluguel por vários portais, à desconfiança causada pelo anonimato online e à falta de verificação da procedência dos anúncios. As limitações das plataformas existentes, que não permitem feedback dos usuários, comprometem ainda mais a confiabilidade.

Diante desse cenário, uma solução viável seria a criação de uma plataforma confiável que agregasse anúncios de aluguel em Florianópolis e na área metropolitana. No entanto, é essencial que essa plataforma permita interações entre os usuários e feedback sobre os anúncios, aumentando assim a confiabilidade e melhorando a experiência do usuário na busca por moradia.

Essa proposta não só aborda a demanda latente por moradia acessível para estudantes da UFSC, mas também se alinha à crescente importância das plataformas digitais na democratização do acesso à informação e na transparência

do mercado imobiliário, respondendo à necessidade de soluções inovadoras para problemas sociais.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

O objetivo geral deste trabalho é a criação de um sistema web que será um agregador de vagas de moradia. Os dados serão coletados utilizando técnicas de *web scraping*. Além disso, ele terá um mecanismo de busca de anúncios com diferentes critérios, uma funcionalidade para fornecimento de *feedback* para os anúncios postados, bem como para sinalização possíveis golpes virtuais. Também será possível receber recomendações de anúncios de aluguel e anúncios de housemates baseados nas buscas realizadas.

1.2.2 OBJETIVOS ESPECÍFICOS

- Expandir o conhecimento sobre modelos de *web scraping.*
- Expandir o conhecimento sobre modelos de recomendação de anúncios.
- Selecionar um modelo de *web scraping* adequado para agregação de anúncios de aluguel.
- Selecionar um modelo de um sistema de recomendação que seja adequado para recomendação de usuários e anúncios afins.
- Desenvolver a aplicação web (*back-end* e *front-end*) de agregação de anúncios de aluguel em Florianópolis.
- Utilizar o AWS para hospedagem da aplicação.

1.2.3 OBJETIVOS OPCIONAIS

- Implementação do modelo de *matching* de usuários com anúncios e com outros outros usuários que queiram dividir o aluguel.
- Avaliar a usabilidade do aplicativo inteligente por meio de um teste de usabilidade.

2 METODOLOGIA

A metodologia de pesquisa utilizada neste trabalho está estruturada nas seguintes etapas:

Etapa 1 – Fundamentação teórica

A revisão da literatura narrativa ou tradicional, quando comparada à revisão sistemática, apresenta uma temática mais aberta; dificilmente parte de uma questão específica bem definida, não exigindo um protocolo rígido para sua confecção; a busca das fontes não é pré-determinada e específica, sendo frequentemente menos abrangente. A seleção dos artigos é arbitrária, provendo o autor de informações sujeitas a viés de seleção, com grande interferência da percepção subjetiva (CORDEIRO, 2007). Nesta etapa são realizadas as seguintes atividades:

A1.1 - Analisar a fundamentação teórica sobre *web scraping.*

A1.2 - Analisar a fundamentação teórica sobre os serviços AWS.

A1.3 - Analisar a fundamentação teórica sobre sistemas de recomendação.

Etapa 2 – Estado da arte

Nesta etapa é realizada a revisão da literatura para identificar e analisar o estado da arte de *web scraping* e sistemas de recomendação. Esta etapa é dividida nas seguintes atividades:

A2.1 – Execução da busca e seleção de artigos relevantes sobre *web scraping*.

A2.2 – Execução da busca e seleção de artigos relevantes sobre sistemas de recomendação.

A2.3 – Extração e análise de informações relevantes sobre *web scraping* e sistemas de recomendação.

Etapa 3 – Desenvolvimento do *back-end* da aplicação

Nesta etapa é desenvolvido todo o *back-end* da aplicação. Esta etapa é dividida nas seguintes atividades:

A3.1 - Análise de requisitos do *back-end*.

A3.2 - Especificação do esquema relacional de dados.

A3.3 - Implementação do modelo de *web scraping.*

A3.4 - Implementação do sistema de recomendação.

A3.5 - Implementação de toda estrutura *back-end* do sistema *web*.

Etapa 4 – Desenvolvimento do *front-end* da aplicação

Nesta etapa é desenvolvido todo o *back-end* da aplicação. Esta etapa é dividida nas seguintes atividades:

A4.1 – Análise de requisitos do *front-end*.

A4.2 – Design de interface.

A4.3 – Implementação de toda estrutura *front-end* do sistema *web*.

A4.4 – Testes do sistema.

Etapa 5 (OPCIONAL) - Implantação da aplicação no serviço AWS

Nesta etapa será realizada a implantação da aplicação na nuvem (abordado na seção 3.2 deste documento) AWS. Esta
etapa é dividida nas seguintes atividades:

A6.1 – Configuração da conta AWS.

A6.2 – Configuração da infraestrutura AWS para o back-end da aplicação.

A6.3 – Configuração da infraestrutura aWS para o front-end da aplicação.

A6.4 – Configuração do CORS para a aplicação.

A6.5 – Testar a aplicação.

Etapa 6 (OPCIONAL) – Avaliação do sistema *web*

Nesta etapa é avaliado o app inteligente por meio de um teste de usabilidade. Esta etapa é dividida nas seguintes atividades:

A5.1 – Definição do teste de usabilidade.

A5.2 – Execução do teste e coleta de dados.

A5.3 – Análise e interpretação dos dados

3 FUNDAMENTAÇÃO TEÓRICA

Para atingir o principal objetivo deste trabalho, o desenvolvimento do sistema TCC UFSC, diversos conceitos foram estudados. Neste capítulo será feita uma breve descrição deles.

3.1 APLICAÇÃO WEB

De acordo com Gellersen (1999), uma aplicação web pode ser definida como um sistema de software que fornece serviços dinâmicos sobre uma infraestrutura distribuída. Ele consiste em um conjunto de componentes que se comunicam entre si por meio de vários protocolos, como HTTP, TCP/IP ou protocolos proprietários. Os componentes podem estar localizados em máquinas diferentes e podem ser implementados em diferentes linguagens de programação. Um aplicativo da Web geralmente fornece uma interface de usuário que é acessada por meio de um navegador da Web, mas também pode fornecer interfaces para outros tipos de clientes, como dispositivos móveis ou outros sistemas de software. Por conta da alta flexibilidade de sua natureza, as aplicações web oferecem uma série de vantagens, como:

ACESSIBILIDADE

As aplicações podem ser acessadas de todos os navegadores da Web e em vários dispositivos pessoais e empresariais por pessoas em diferentes locais.

DESENVOLVIMENTO EFICIENTE

O processo de desenvolvimento de aplicações Web é relativamente simples e econômico para as empresas, já que times pequenos podem realizar ciclos de desenvolvimento curtos, tornando as aplicações um método eficiente e acessível de criar programas de computador. Além disso, a mesma versão funciona em todos os navegadores e dispositivos modernos na grande maioria dos casos, não sendo mais necessário criar várias iterações diferentes para várias plataformas.

SIMPLICIDADE PARA O USUÁRIO

As aplicações na nuvem dispensam a necessidade de downloads convencionais, simplificando o acesso e eliminando preocupações com manutenção e espaço em disco para o usuário. Elas recebem atualizações automáticas de software e segurança, garantindo que estejam sempre na versão mais recente, reduzindo consideravelmente os riscos de violações de segurança. Apesar da existência de infraestrutura adicional e hardware custoso, esses aspectos, quando na nuvem, não são uma preocupação direta para a empresa..

ESCALABILIDADE

As empresas que usam aplicações Web podem adicionar usuários conforme necessário, sem infraestrutura adicional ou hardware caro. Além disso, a grande maioria dos dados das aplicações Web é armazenada na nuvem, o que significa que sua empresa não precisará investir em capacidade de armazenamento adicional para executá-las.

3.2 COMPUTAÇÃO NA NUVEM

Segundo Taurion (2009), o termo “computação em nuvem” surgiu em 2006 em uma palestra de Eric Schmidt, da Google, sobre como a empresa gerenciava seus *data centers* (local onde são concentrados os computadores e sistemas responsáveis pelo processamento de dados de uma empresa ou organização). Atualmente a computação em nuvem se apresenta como o cerne de um movimento de profundas transformações do mundo da tecnologia (TAURION, 2009). A palavra nuvem, neste caso, é uma representação para a Internet ou infraestrutura de comunicação entre os componentes arquiteturais, baseada na abstração de infraestrutura. Cada parte desta infraestrutura é provida como um serviço e, estes serviços são normalmente alocados em data centers, utilizando hardware compartilhado para computação e armazenamento (SOUSA; MOREIRA; MACHADO, 2009). A computação em nuvem é um novo modelo de computação que move todos os dados e as aplicações dos usuários para grandes centros de armazenamento. Com isso, as aplicações e os sistemas de hardware são distribuídos na forma de serviços baseados na Internet. Fundamentada em conceitos já estabelecidos previamente, como a virtualização e o modelo

pay-per-use (modelo de pagamento baseado no uso, semelhante aos serviços de telefonia e energia elétrica), a computação em nuvem possui uma série de vantagens, como a possibilidade de ampliar os recursos utilizados sempre que necessário (SANTOS; PEDROSA; SANTOS; 2009).

3.3 PLATAFORMA COMO SERVIÇO

Platform as a Service (*PaaS*) é a ideia de que alguém pode fornecer o hardware (como em *IaaS*) mais uma certa quantidade de software de aplicativo - como a integração em um conjunto comum de funções de programação ou bancos de dados como base sobre a qual você pode criar seu aplicativo. O *PaaS* é uma plataforma de desenvolvimento e implantação de aplicativos fornecida como um serviço para desenvolvedores na Web. Ele facilita o desenvolvimento e a implantação de aplicativos sem o custo e a complexidade de comprar e gerenciar a infraestrutura subjacente, fornecendo todas as facilidades necessárias para dar suporte ao ciclo de vida completo de criação e entrega de aplicativos e serviços da Web totalmente disponíveis na Internet. Essa plataforma consiste em software de infraestrutura e normalmente inclui um banco de dados, middleware e ferramentas de desenvolvimento. Uma arquitetura de computação em grade virtualizada e em cluster costuma ser a base desse software de infraestrutura. Alguns

As ofertas de PaaS têm uma linguagem de programação ou API específica. Por exemplo, o Google AppEngine é uma oferta de PaaS em que os desenvolvedores escrevem em Python ou Java. EngineYard é Ruby on Rails. Às vezes, os provedores de PaaS têm linguagens proprietárias, como force.com da Salesforce.com e Coghead, agora propriedade da SAP (BHARDWAJ; JAIN; JAIN; 2010).

3.4 AMAZON WEB SERVICES

É um conjunto de serviços de infraestrutura de TI para empresas na forma de serviços da web – agora comumente conhecidos como computação em nuvem. Um dos principais benefícios da computação em nuvem é a oportunidade de substituir as despesas iniciais de infraestrutura de capital por custos variáveis baixos que se adaptam ao seu negócio. Com a nuvem, as empresas não precisam mais planejar e

adquirir servidores e outras infraestruturas de TI com semanas ou meses de antecedência. Em vez disso, eles podem ativar instantaneamente centenas ou milhares de servidores em minutos e fornecer resultados mais rapidamente. A AWS fornece uma plataforma de infraestrutura altamente confiável, escalável e de baixo custo na nuvem que capacita centenas de milhares de empresas em 190 países ao redor do mundo (MATHEW; VARIA; 2014). Os principais serviços da AWS que compõem a plataforma são:

3.4.1 AMAZON EC2

EC2 é um serviço de computação em nuvem que permite a criação e gerenciamento de instâncias de máquinas virtuais para executar o seu código de back-end em um ambiente escalável e flexível. A utilização do Amazon EC2 possui uma camada gratuita que permite a utilização de 750 horas de instâncias t2.micro por mês durante um ano. As instâncias podem ser configuradas com diferentes tipos de instâncias, tamanhos de recursos computacionais (*CPU, RAM*, armazenamento), sistemas operacionais e outras configurações personalizadas, de acordo com as necessidades da aplicação. O deploy de uma aplicação web no Amazon EC2 pode ser realizado de várias maneiras, como por exemplo, criar uma imagem personalizada da aplicação e implantá-la em uma instância do EC2. No entanto, é importante ressaltar que a escolha da melhor abordagem e configuração dependerá dos requisitos específicos da aplicação, da carga de trabalho, dos recursos necessários e de outros fatores relevantes (MATHEW; VARIA; 2014).

3.4.2 AMAZON API GATEWAY

O Amazon API Gateway é outro componente do arsenal sem servidor da AWS. Ele é usado para criar e gerenciar APIs RESTful e WebSocket na frente de sua funcionalidade de back-end. Existem vários motivos para usar um API Gateway, e o primeiro é abstrair a implementação de seu aplicativo ou serviço do cliente. Isso permite que você tenha maior flexibilidade em como a lógica e o processamento de negócios são criados e para que o cliente não precise entender as estruturas de dados subjacentes ou a camada de armazenamento. Outro motivo para um componente de gateway é descarregar as responsabilidades de autenticação,

autorização e gerenciamento de certificados. Isso pode aliviar a carga em um microsserviço e permitir que a implementação seja reutilizada para outros microsserviços. Isso também ajuda o desenvolvedor a se concentrar na criação da lógica de negócios necessária para o serviço. Um gateway de API, e especialmente o Amazon API Gateway, pode proteger seu serviço contra ataques devido à forma como foi projetado. O Amazon CloudFront fica na frente do API Gateway para filtrar solicitações e distribuir o tráfego entre os pontos de presença. Um dos benefícios de aproveitar o CloudFront é que ele também possui o AWS Shield integrado como padrão. O Shield protege contra ataques distribuídos de negação de serviço (DDoS) das camadas 3 e 4 (PATTERSON, 2019).

3.4.3 AWS LAMBDA

É um serviço de computação sem servidor que pode ser utilizado para executar o código de back-end de forma escalável e flexível. Através dele, os desenvolvedores podem criar funções (trechos de código) em várias linguagens de programação, como Python, Node.js, Java, C# e Go, e configurá-las para serem acionadas automaticamente em resposta a eventos, como chamadas de API, upload de arquivos, alterações em bancos de dados, entre outros. Quando um evento ocorre, o código da função é executado e os recursos necessários são provisionados automaticamente, sem a necessidade de configurar servidores ou escalar a infraestrutura manualmente. Além disso, este serviço oferece escalabilidade automática, o que significa que a quantidade de recursos alocados para a função é ajustada automaticamente com base na demanda, garantindo alta disponibilidade e performance. Finalmente, o AWS Lambda possui uma camada gratuita que permite a execução de até 1 milhão de solicitações e 400.000 GB-segundos de tempo de computação por mês durante um ano (AMAZON WEB SERVICES, 2023).

3.4.4 AWS CLOUDFORMATION

É um serviço de infraestrutura como código que permite a criação e gerenciamento de recursos da AWS de forma automatizada. O AWS CloudFormation utiliza modelos para definir a infraestrutura como código. Os

modelos são arquivos de texto escritos em formato JSON ou YAML (YAML Ain't Markup Language) que descrevem os recursos que você deseja criar, suas propriedades e suas configurações. Os modelos do AWS CloudFormation são reutilizáveis e podem ser versionados, o que facilita o gerenciamento do código de infraestrutura. O AWS CloudFormation é gratuito, mas os recursos criados por meio dele são cobrados(AMAZON WEB SERVICES, 2023).

3.5 WEB SCRAPING

A raspagem da Web, também conhecida como extração ou colheita da Web, é uma técnica para extrair dados da World Wide Web (WWW) e salvá-los em um sistema de arquivos ou banco de dados para posterior recuperação ou análise. Normalmente, os dados da web são descartados utilizando o Protocolo de Transferência de Hipertexto (HTTP) ou por meio de um navegador da web. Isso é feito manualmente por um usuário ou automaticamente por um bot ou rastreador da web. Devido ao fato de que uma enorme quantidade de dados heterogêneos é constantemente gerada na WWW, o web scraping é amplamente reconhecido como uma técnica eficiente e poderosa para coletar *big data* (ZHAO, 2001). O web scraping torna rápido e direto o acesso à grande quantidade de informações disponíveis online. Em comparação com a extração manual de dados de sites, é muito mais rápido e fácil. Hoje em dia, a raspagem da web é cada vez mais comum. Muita coleta de dados e extração de informações podem ser feitas de forma rápida e fácil usando um software de extração de dados online. Em comparação com a extração manual de dados de sites, é muito mais rápido e fácil. Hoje em dia, a raspagem da web é cada vez mais comum. Muita coleta de dados e extração de informações podem ser feitas de forma rápida e fácil usando um software de extração de dados online. No entanto, quando os indivíduos usam o termo "web scrapers", geralmente se referem a programas de computador. O software de raspagem da Web (às vezes conhecido como "bots") é projetado para navegar em sites, raspar as páginas pertinentes e extrair dados significativos. Esses bots podem recuperar rapidamente enormes volumes de dados automatizando esse processo. Na era digital, quando o big data desempenha um papel tão importante e está em constante atualização e mudança, isso tem vantagens óbvias. A raspagem da Web tem vários usos, principalmente na área de análise de dados (PERSSON, 2019).

3.5.1 ETAPAS DE WEB SCRAPING

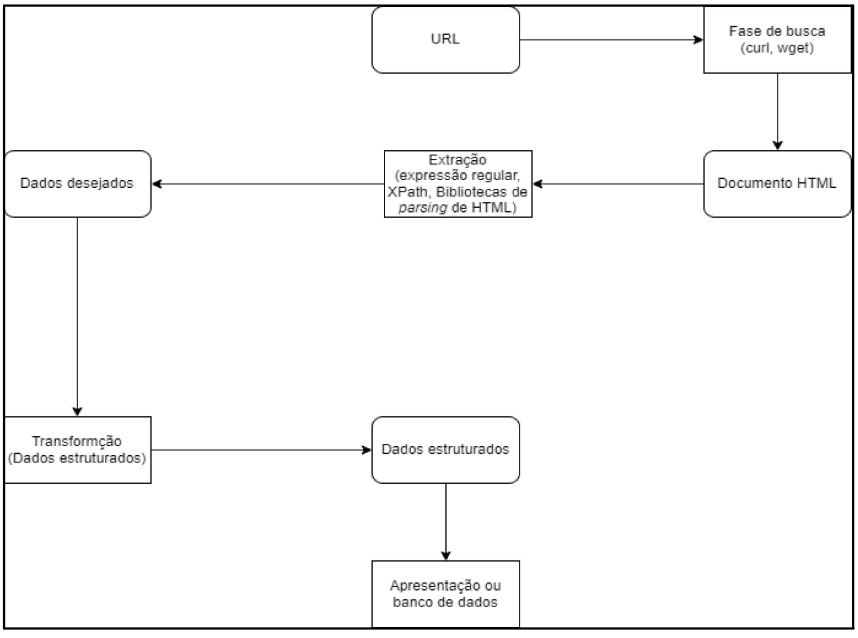
O processo de web scraping, de maneira geral, pode ser separado em 3 etapas, conforme mostrado na fig.1, as quais são:

Fase de busca: O site desejado com as informações relevantes deve primeiro ser acessado no que é conhecido como fase de busca. Isso é feito por meio do protocolo HTTP, que é um protocolo da Internet para enviar e receber solicitações de servidores da Web. Os navegadores da Web utilizam métodos semelhantes para obter material em páginas da Web. Nesta etapa, bibliotecas como curl 2 e wget 3 podem ser usadas enviando uma solicitação HTTP GET para o endereço de destino (URL) e obter a página HTML como resposta (PERSSON, 2019).

Fase de extração: Depois de recuperar a página HTML, os dados importantes devem ser extraídos. Expressões regulares, bibliotecas de análise de HTML e consultas XPath são utilizadas nesta etapa, que é conhecida como estágio de extração. O XPath é uma ferramenta para encontrar informações em documentos (PERSSON, 2019).

Fase de transformação: Uma vez que restam apenas os dados relevantes, eles podem ser convertidos em um formato estruturado para apresentação ou armazenamento. Usando os dados armazenados, é possível coletar informações que podem ajudar a inteligência de negócios na tomada de uma decisão melhor (PERSSON, 2019).

Figura 1. Processo de Web Scraping



3.5.2 TECNOLOGIAS DE WEB SCRAPING

Existem inúmeras técnicas de *web scraping* que podem ser utilizadas para coleta de dados. Cada uma tem uma aplicação dentro de um ou vários cenários específicos. As mais populares são:

EXPRESSÃO REGULAR

Expressões regulares, conhecidas como regex, são elementos cruciais para a manipulação eficiente de strings em JavaScript. Elas oferecem um método flexível e poderoso para buscar, substituir e validar padrões em textos, contribuindo significativamente para o desenvolvimento de aplicações robustas.

A criação de expressões regulares em JavaScript pode ser realizada de duas maneiras principais. Utilizando a classe *RegExp* ou a sintaxe literal entre barras (/), a escolha dependerá da preferência do desenvolvedor e da situação específica.

O JavaScript disponibiliza diversos métodos para trabalhar com expressões regulares, como por exemplo, o método *test()* verifica se a expressão regular encontra um padrão na string, retornando verdadeiro ou falso com base nessa verificação. Já o método *exec()* fornece detalhes sobre a primeira correspondência encontrada ou null se nenhuma correspondência for identificada. Outro método útil é o *match()*, utilizado para obter um *array* contendo todas as correspondências encontradas na string, permitindo um processamento posterior.

Além disso, os modificadores são sufixos adicionados à expressão regular que modificam o comportamento da busca. O modificador g (global) permite encontrar todas as ocorrências, i (case-insensitive) ignora a diferença entre maiúsculas e minúsculas, e m (multiline) possibilita a busca por correspondências em várias linhas (MDN, 2023).

CHEERIO

Cheerio representa uma biblioteca JavaScript altamente eficaz, especialmente concebida para análise e manipulação de dados HTML no ambiente Node.js. Destacando-se por sua agilidade e eficiência, a biblioteca tornou-se uma escolha popular para desenvolvedores envolvidos em tarefas como web scraping, extração de dados e manipulação de documentos HTML de forma programática.

O funcionamento do Cheerio é profundamente influenciado pela sintaxe do jQuery, o que facilita a transição para aqueles familiarizados com essa biblioteca. Através do Cheerio, os desenvolvedores podem empregar seletores familiares do jQuery para localizar e manipular elementos HTML em documentos. Isso oferece uma abordagem mais intuitiva e simplificada para navegar pelo DOM.

Uma característica notável do Cheerio é sua capacidade de operar diretamente no lado do servidor, eliminando a necessidade de um navegador. Isso torna a biblioteca ideal para projetos Node.js, nos quais a interação com um ambiente de navegador é inviável ou indesejada. Ao realizar operações de parsing HTML de forma eficiente, o Cheerio permite que os desenvolvedores extraiam dados específicos de uma página web com facilidade, contribuindo para a automação de processos de coleta de informações.

Além disso, o Cheerio suporta uma ampla gama de funcionalidades, como manipulação de atributos, adição e remoção de elementos, e navegação avançada pelo DOM. Essa flexibilidade faz da biblioteca uma ferramenta versátil para lidar com uma variedade de cenários de análise de dados na web.

Em resumo, o Cheerio não apenas oferece uma solução eficaz para tarefas de scraping, mas também proporciona uma experiência de desenvolvimento mais intuitiva e eficiente para aqueles que buscam analisar e manipular dados HTML de maneira programática no ambiente Node.js (CHEERIO, 2023).

3.6 SISTEMAS RECOMENDADORES

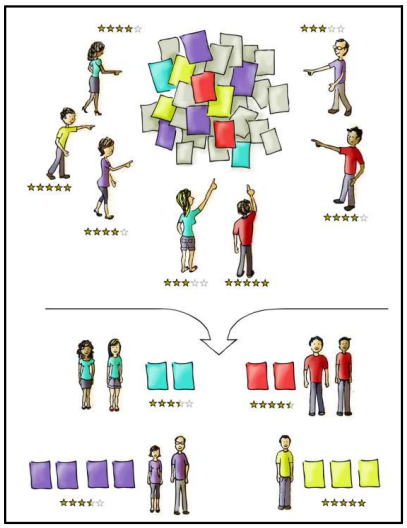
Sistemas de recomendação são filtros de informação para apresentar itens ou objetos – como páginas web, filmes, músicas, livros, medicamentos, lojas, artigos – que provavelmente são do interesse do usuário. O princípio dos sistemas de recomendação se baseia em “o que é relevante para mim, também pode ser relevante para alguém com interesse similar”. A grande maioria das técnicas de recomendação dependem da avaliação da informação realizada pelos indivíduos DA MOTTA, 2012). Eles são caracterizados de acordo com três eixos:

- Tipo de Entrada e Saída
- Itens do Projeto
- Método de Recomendação

Um Sistema de Recomendação é um sistema colaborativo porque a recomendação é feita a partir da organização, manipulação, sumarização e agrupamento das avaliações individuais. O indivíduo contribui para avaliação de produtos que serão consumidos por outros indivíduos. É um processo coletivo, embora a interação seja assíncrona e os benefícios não sejam percebidos no momento da contribuição individual ao grupo.

Os principais componentes de um sistema de recomendação são cliente e produto. Um produto é um recurso que pode ser de diferentes naturezas, por exemplo: um conteúdo, um arquivo, uma informação, uma pessoa, um objeto. A recomendação é uma função de mapeamento de interesses do cliente para obtenção de um ou mais produtos. Todos nós já nos deparamos com sugestões intercaladas às informações que acessamos na rede, como pode ser visto quando fazemos compras em algum site de e-commerce, conforme ilustrado na Figura 2.

Figura 2 Sistema de recomendação de e-commerce



3.7 REST API

A API REST representa um conjunto de recursos acessíveis por meio de caminhos relativos ao caminho base. Esses caminhos, muitas vezes hierárquicos, definem os recursos disponíveis na API e uma estrutura bem projetada facilita a compreensão dos recursos disponíveis. Por exemplo, em um banco de dados de alunos, os recursos podem incluir operações como obter informações de alunos, adicionar um novo aluno ou excluir um aluno existente. Cada recurso na API possui

operações específicas, identificadas por um nome e um método HTTP (como GET, POST ou DELETE). Esses métodos são únicos por operação e são combinados com os caminhos para identificar precisamente o recurso e a operação requisitada pelo cliente HTTP (SURWASE, 2016).

3.8 JQUERY

A biblioteca *jQuery* fornece uma camada de abstração de uso geral para scripts da Web e, portanto, é útil em uma ampla gama de contextos online, como interações de interface do usuário, manipulação de eventos, requisições *AJAX*, animações e muito mais. Sua utilidade geral a torna vantajosa em praticamente todas as situações que envolvem a criação e aprimoramento de dinâmicas e funcionalidades em páginas web. Os recursos principais nos ajudam a realizar as seguintes tarefas:

- Acessar elementos em um documento: sem uma biblioteca *JavaScript*, os desenvolvedores da Web geralmente precisam escrever muitas linhas de código para percorrer a árvore do *DOM* e localizar partes específicas da estrutura de um documento *HTML*. Com o *jQuery*, os desenvolvedores têm à sua disposição um mecanismo seletor robusto e eficiente, facilitando a recuperação da parte exata do documento que precisa ser inspecionada ou manipulada.

- Modificar a aparência de uma página da *Web*: o *CSS* oferece um método poderoso de determinar a maneira como um documento é renderizado, mas fica aquém quando nem todos os navegadores da *Web* suportam os mesmos padrões. O *jQuery* capacita os desenvolvedores a superar inconsistências entre navegadores, proporcionando um suporte consistente aos padrões da web em todas as plataformas. Sua eficácia reside na capacidade de modificar classes ou propriedades de estilo de forma consistente em diferentes navegadores, mesmo após a renderização da página. Isso significa que, independentemente do navegador utilizado, o *jQuery* oferece uma abordagem uniforme na aplicação de mudanças no design e no comportamento de elementos específicos do documento, reduzindo significativamente a dependência de compatibilidade entre plataformas.

- Alterar o conteúdo de um documento: não se limitando a meras mudanças cosméticas, o jQuery pode modificar o conteúdo de um documento em si com apenas algumas teclas. O texto pode ser alterado, as imagens podem ser inseridas

ou trocadas, as listas podem ser reordenadas ou toda a estrutura do *HTML* pode ser reescrita e estendida - tudo com uma *API* fácil de usar.

- Responder à interação de um usuário: mesmo os comportamentos mais elaborados e poderosos não são úteis se não pudermos controlar quando eles ocorrem. A biblioteca *jQuery* oferece uma maneira elegante de interceptar uma ampla variedade de eventos, como um clique do usuário em um link, sem a necessidade de sobrecarregar o próprio código *HTML* com manipuladores de eventos. Ao mesmo tempo, sua *API* de manipulação de eventos remove as inconsistências do navegador que geralmente afligem os desenvolvedores da web.

- Recuperar informações de um servidor sem atualizar uma página: esse padrão de código é conhecido como *AJAX*, que originalmente significava *JavaScript* e *XML* assíncronos, mas desde então passou a representar um conjunto muito maior de tecnologias para comunicação entre o cliente e o servidor. A biblioteca *jQuery* remove a complexidade específica do navegador desse processo responsivo e rico em recursos, permitindo que os desenvolvedores se concentrem na funcionalidade final do servidor (CHAFFER, 2013).

3.9 MODELO MVC

A sigla *MVC* significa *Model-View-Controller* e é um padrão arquitetural amplamente utilizado no desenvolvimento de software. Esse padrão sugere uma arquitetura de software dividida em componentes, viabilizando com clareza o desenvolvimento de um código organizado e enxuto, e posteriormente, a reciclagem e manutenção do sistema sem dificuldade e com segurança (LUCIANO; ALVES; 2017). Porém, a independência dos componentes só será atingida se houver uma organização do sistema em camadas para garantir a escalabilidade, eficiência e a reusabilidade.

No padrão *MVC*, o Modelo trabalha na manipulação dos dados internos de uma aplicação, e se comunica especialmente com o armazenamento de dados. A camada de Visão ou apresentação trabalha na interface do usuário, capturando as suas ações e enviando ao Controlador, acessa os dados do Modelo através do Controlador e aplica a apresentação desses dados conforme o evento. Por fim, a camada de Controle é responsável por gerenciar as interações entre a camada de Visão e o Modelo, controlando os fluxos de dados e a lógica da aplicação. Além

disso, esta camada coordena as ações do usuário, processa as requisições recebidas, e formula as respostas para apresentação ou manipulação dos dados pelo Modelo. (LUCIANO; ALVES; 2017).

3.10 MONGODB

O *MongoDB* é um banco de dados de documentos de código aberto que fornece alto desempenho, alta disponibilidade e dimensionamento automático. Um registro no MongoDB é um documento, que é uma estrutura de dados composta por pares de campos e valores. Os documentos do *MongoDB* são semelhantes aos objetos *JSON*. Os valores dos campos podem incluir outros documentos, arrays e arrays de documentos. As vantagens de usar documentos são:

- Documentos correspondem a tipos de dados nativos em muitas linguagens de programação.
- Documentos e matrizes incorporados reduzem a necessidade de junções caras vistas no MySQL.

O *MongoDB* é uma opção bastante utilizada para armazenamento de dados pois é fácil de aprender e mais rápido do que os bancos de dados relacionais tradicionais. Além disso, ele é desnormalizado, o que significa que os dados são armazenados em uma estrutura de documento aninhada em vez de tabelas relacionais.(CHAUHAN, 2019).

3.10 NODE.JS

O *Node* é uma plataforma criada a partir de um *port* da *engine* do *Chrome* chamada V8 e é muito utilizado para criar facilmente aplicativos *web* escalonáveis e rápidos. Ele usa um modelo de E/S sem bloqueio e orientado a eventos que o torna leve e eficiente, perfeito para aplicativos em tempo real com uso intensivo de dados executados em dispositivos distribuídos (CANTELON, 2014).

Uma caraterística muito importante *Node* são as *Promises*, as quais são objetos que representam o sucesso ou a falha eventual (MDN WEB DOCS, 2023) de uma operação assíncrona. Elas oferecem um mecanismo mais estruturado para lidar com fluxos assíncronos, permitindo encadear operações de forma clara. Além disso, a utilização de *async/await* simplifica ainda mais o código assíncrono,

tornando-o semelhante à programação síncrona. Isso melhora a legibilidade e a manutenção do código, ao mesmo tempo que preserva a eficiência inerente ao modelo de E/S sem bloqueio. Ao incorporar essas duas funcionalidades em projetos *Node.js*, é possível aproveitar os benefícios de um código mais conciso e compreensível, evitando a complexidade associada a *callbacks* aninhados. Essas funcionalidades promovem uma abordagem mais moderna e eficiente para lidar com operações assíncronas, contribuindo para a construção de aplicativos robustos e responsivos na plataforma *Node.js.*

3.12 EXPRESS.JS

O *Express.js* é uma biblioteca baseada no módulo principal *Node.js HTTP* e nos componentes *Connect*. Ele simplifica a criação e gerenciamento das *APIs* do *node.js* e adiciona novos recursos úteis a elas. Os componentes Connect anteriormente mencionados também são chamados de *middlewares*. Eles são o cerne da filosofia do *framework*, ou seja, configuração sobre convenção. Em outras palavras, os desenvolvedores são livres para escolher as bibliotecas de que precisam para um projeto específico, o que lhes proporciona flexibilidade e alta personalização (MARDAN, 2014).

3.13 JSON WEB TOKEN

O *JWT* é um padrão amplamente utilizado para transmitir informações entre diferentes entidades de forma compacta e independente. Ele é comumente empregado como um token de acesso em sistemas de autenticação e autorização na web, composto por três partes: cabeçalho, payload e assinatura. Cada parte é codificada em Base64 e unida para formar o token completo.

No cabeçalho, são especificados o tipo do *token* e o algoritmo de assinatura utilizado. O *payload* contém as reivindicações, declarações sobre uma entidade (normalmente o usuário) e metadados adicionais, como nome de usuário, papéis e prazo de validade. A assinatura, gerada a partir do cabeçalho, *payload* e uma chave secreta, assegura a integridade do *token*.

A grande vantagem do *JWT* é sua verificabilidade e confiabilidade, já que somente a parte que possui a chave secreta pode validar a assinatura. Além disso,

sua estrutura compacta facilita a transmissão via *URLs*, parâmetros de consulta e cabeçalhos *HTTP*. Utilizando *JWTs*, as aplicações podem implementar autenticação stateless, pois as informações necessárias estão contidas no próprio token, reduzindo a necessidade de consultas frequentes ao servidor para verificar a autenticidade do usuário e melhorando o desempenho.

Para invalidar uma sessão autenticada usando *JWT*, uma abordagem comum é manter uma lista de tokens inválidos ou revogados no servidor. Se um usuário fizer *logout* ou se for necessário invalidar a sessão por qualquer motivo, o servidor pode adicionar o token correspondente a uma lista de revogação. Quando um cliente tenta usar um token, o servidor verifica se ele está na lista de tokens inválidos antes de aceitá-lo como válido. Essa prática é útil para garantir que mesmo os tokens previamente válidos não sejam mais aceitos após serem revogados, garantindo a segurança do sistema (RFC 7519, 2023).

4 ESTADO DA ARTE

Nesta etapa, será realizada a síntese de estudos de pesquisa anteriores, artigos científicos, teses, dissertações, relatórios técnicos e outras fontes relevantes de literatura acadêmica. Foram selecionados cinco documentos científicos (artigos, periódicos, capítulos de livros, etc.) a respeito das tecnologias que serão empregadas no desenvolvimento da aplicação: um artigo a respeito de *web scraping*, um estudo comparativo entre os principais frameworks para desenvolvimento do front-end de aplicações web e três artigos que abordam os variados aspectos de sistemas de recomendação.

4.1 TÍTULO: “WEB SCRAPING: APPLICATIONS AND SCRAPING TOOLS” AUTORES: PRIYA MATTA, NIKITA SHARMA, DEVYANI SHARMA, BHASKER PANTE E SACHIN SHARMA.

O texto aborda a importância do web scraping como uma técnica para extrair dados de websites de forma automatizada, especialmente no contexto do e-commerce. Os autores destacam como o enorme volume de dados disponíveis na era atual, em diferentes formatos (texto, imagem, áudio e vídeo) torna o web scraping uma ferramenta valiosa para coletar informações de concorrentes, analisar padrões de compra dos clientes e obter dados relevantes para os negócios. Além disso, são apresentadas várias definições de web scraping propostas por diferentes acadêmicos e profissionais, destacando como o termo é usado de forma intercambiável com outros conceitos como *web extraction*, *web harvesting* e *data mining*. O objetivo principal do *web scraping* é extrair informações de websites e salvá-las em formato de arquivo ou banco de dados para posterior recuperação e análise.

O texto também discute a motivação por trás do uso do web scraping, enfatizando a necessidade de coletar grandes quantidades de dados de forma eficiente e automatizada, uma vez que o processo manual de copiar e colar informações não é viável em websites com centenas de páginas. Os autores destacam a aplicação do web scraping em diversas áreas, como a obtenção de

dados sobre preços de ações e contatos de empresas, e como essa técnica pode simplificar a tarefa de extração de dados de forma mais fácil e rápida.

O artigo menciona também algumas ferramentas disponíveis para realização de *web scraping*, destacando formatos comuns como CSV e Excel, bem como formatos mais avançados como JSON, o qual é vantajoso por ser um formato muito utilizado em APIs. Os autores concluem destacando a importância do *scraping* como uma abordagem amplamente utilizada para a extração de dados da web, embora também enfrente algumas controvérsias em relação à legalidade e ética da coleta de dados em alguns websites.

4.2 TÍTULO: "COMPARATIVE ANALYSIS ON FRONT-END FRAMEWORKS FOR WEB APPLICATIONS"
AUTOR: RISHI VYAS

O texto compara três frameworks populares de desenvolvimento web: Angular, React e Vue. O Angular é considerado uma "plataforma" devido ao seu amplo suporte a diversas funcionalidades, como controle de interface, validação de formulários, roteamento, gerenciamento de estado e teste de ponta a ponta. O React é uma biblioteca conhecida pela reutilização de código, facilidade de uso e eficiência, utilizando um DOM virtual para renderização sob demanda. Já o Vue.js fica entre o Angular e o React em termos de tamanho e recursos, destacando-se por seu desempenho, gerenciamento de estado embutido e sistema de roteamento.

O Angular se destaca por sua amplitude de funcionalidades e atualizações frequentes, utilizando TypeScript e fornecendo ferramentas internas para traduzir seu código em JavaScript. O React, por sua vez, é elogiado pela reutilização de código, facilidade de customização e eficiência, utilizando um DOM virtual. O Vue.js é reconhecido por seu desempenho, gerenciamento de estado embutido e sistema de roteamento, utilizando HTML, JS e CSS separadamente.

O artigo compara os frameworks em termos de popularidade e desempenho, destacando o Angular como robusto e testado, o React como flexível e rápido, e o Vue.js por sua simplicidade e alto desempenho. O Vue.js é identificado como um framework em crescimento, enquanto o Angular mantém sua posição com um crescimento mais lento. No geral, os três frameworks são reconhecidos por suas

vantagens e podem ser escolhidos com base nos requisitos e objetivos específicos de cada aplicação.

4.3 TÍTULO: “A SURVEY ON RECOMMENDATION SYSTEM”
AUTORES: DEBASHIS DAS, LAXMAN SAHOO E SUJOY DATTA

O artigo discute em detalhes os sistemas de recomendação (RS), destacando diferentes tipos e algoritmos populares, com ênfase especial no sistema de filtragem baseado em conteúdo. Esse método sugere itens com base na similaridade com itens anteriores ou em exibição, não dependendo de dados de outros usuários e evitando problemas como a inicialização a frio.

A importância dos RS é explorada em diversos setores, como comércio eletrônico, mídias sociais e notícias, com exemplos de empresas como a Amazon. Outros tipos de RS, como filtragem colaborativa, abordagens híbridas e com reconhecimento de contexto, são discutidos.

A revisão de literatura inclui referências relevantes, como um estudo sobre recomendação de viagens a partir de fotos da comunidade, destacando contribuições e desafios de privacidade. Outros artigos abordam métodos eficientes de recomendação em vídeos e serviços, considerando aspectos como escalabilidade e precisão, mas também mencionam complexidade de implementação.

Uma técnica inovadora de agrupamento baseada no modelo LCRM é apresentada para e-commerce, considerando classificações gerais e valores de opinião. Outros estudos propõem recomendações baseadas em localização e combinam preferências e opiniões do usuário para melhorar a precisão.

O artigo conclui ressaltando a crescente importância dos RS na era do big data, destacando seu potencial para marketing de precisão e maximização de lucros em diversas áreas.

4.4 TÍTULO: “RECOMMENDATION SYSTEMS: ALGORITHMS, CHALLENGES, METRICS, AND BUSINESS OPPORTUNITIES”
AUTORES: ZESHAN FAYYAZ, MAHSA EBRAHIMIAN, DINA NAWARA, AHMED IBRAHIM E RASHA KASHEF

O capítulo aborda o papel fundamental dos sistemas de recomendação (RSs) em diversas aplicações diante da explosão de informações online. RSs são amplamente utilizados em setores como e-commerce, saúde, transporte e agricultura, visando personalizar a experiência do usuário. O texto destaca categorias de RSs, como filtragem colaborativa e baseada em conteúdo, discutindo suas vantagens e desvantagens, além de abordar o desafio do "início frio" e a escassez de dados.

Os autores discutem problemas recorrentes nos RSs, como o "cold start," e apresentam soluções baseadas em modelos Bayesianos. A escassez de dados é abordada, destacando técnicas como a modelagem das preferências dos usuários com base em comportamentos e conexões sociais. Desafios como escalabilidade, precisão e diversidade são discutidos, com ênfase em técnicas avançadas, filtragem híbrida e métricas de avaliação.

O comércio eletrônico é mencionado como a pioneira na adoção de RSs, com exemplos notáveis como a Amazon, enquanto a agricultura, saúde, transporte e mídia também se beneficiam desses sistemas. As aplicações em mídia incluem museus, aplicativos móveis, recomendações de filmes e música em plataformas como Netflix e YouTube. A influência das redes sociais na personalização de recomendações e as futuras direções de pesquisa, como o desenvolvimento de novas abordagens e considerações de privacidade, concluem o texto.

4.5 TÍTULO: “CONTENT-BASED RECOMMENDER SYSTEMS: STATE OF THE ART AND TRENDS”
AUTORES: PASQUALE LOPS, MARCO DE GEMMIS, GIOVANNI SEMERARO

O texto aborda a importância dos sistemas de recomendação baseados em conteúdo diante do grande volume de informações disponíveis na Web e em bibliotecas digitais. Esses sistemas utilizam algoritmos para analisar documentos e construir perfis de interesses do usuário, facilitando a busca personalizada. O texto

destaca os paradigmas de sistemas de recomendação, baseados em conteúdo e colaborativos, detalhando o processo de recomendação baseado em conteúdo em três etapas: Analisador de Conteúdo, Aprendiz de Perfil e Componente de Filtragem. Diversos tipos de feedback de relevância são discutidos, incluindo explícito e implícito, com diferentes abordagens para construir perfis de usuários.

O estado da arte dos sistemas de recomendação baseados em conteúdo é explorado, destacando a pesquisa em Recuperação de Informação e Inteligência Artificial. A representação semântica, utilizando ontologias, é apresentada como uma abordagem inovadora para lidar com a ambiguidade da linguagem natural. O texto menciona modelos tradicionais, como o espaço vetorial baseado em palavras-chave, e sistemas conhecidos na literatura.

Algoritmos, incluindo Naive Bayes, árvores de decisão e vizinho mais próximo, são discutidos em relação aos sistemas de recomendação. O texto aborda a importância do conteúdo gerado pelo usuário, como folksonomias, e destaca a necessidade de explorar sua integração nos algoritmos de recomendação. A problemática da superespecialização nas recomendações baseadas em conteúdo é abordada, sugerindo a introdução de serendipidade para diversificar as recomendações. Estratégias, como o papel do acaso, princípio de Pasteur, anomalias e exceções, e raciocínio por analogia, são discutidas como maneiras de introduzir elementos surpreendentes nas recomendações.

Em conclusão, o texto destaca a evolução dos consumidores de passivos para contribuintes ativos de informações e como isso influenciou a personalização em sistemas de recomendação. Estratégias adaptativas, como a incorporação de léxicos e folksonomias definidos pelo usuário, são consideradas, e a necessidade de superar a superespecialização é enfatizada.

5 DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo, descreve-se, de maneira geral, a metodologia de desenvolvimento empregada na aplicação. Primeiramente, foram levantados os seguintes requisitos funcionais:

- A aplicação deverá conter pelo menos um *web scraper* para coletar os anúncios de aluguel de imobiliárias na região de Florianópolis.

- A aplicação deverá mostrar todos os anúncios coletados pelo *web scraper*.

- A aplicação deverá permitir que os usuários se cadastrem nela.

- A aplicação deverá permitir que os usuários excluam seus perfis, se assim desejarem.

- A aplicação deverá implementar um algoritmo de recomendação.

- A aplicação deverá conter um botão que exiba recomendações de anúncios de aluguel quando clicado.

- A aplicação deverá permitir que novos usuários preencham suas preferências de listagem durante o cadastro.

- A aplicação deverá permitir que usuários editem seus registros para alterar as preferências de listagem.

- A aplicação deverá fornecer recomendações de listagem com base nas preferências cadastradas nos perfis dos usuários.

Além disso, foram identificados os seguintes requisitos não funcionais:

- A aplicação deverá ser desenvolvida utilizando o padrão MVC sempre que possível.

- A aplicação deverá ser desenvolvida usando o Express.

- A aplicação deverá ser desenvolvida utilizando React no front-end.

- A aplicação deverá ser desenvolvida utilizando Node.js no back-end.

- A aplicação deverá ser implantada no serviço AWS da Amazon.

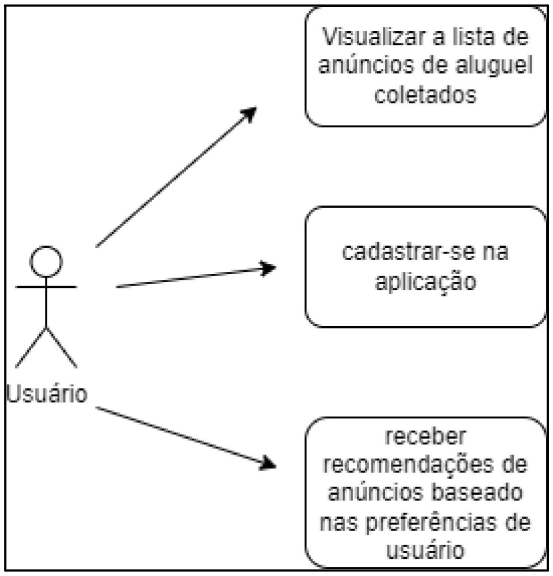
- A aplicação deverá utilizar o banco de dados Mongodb para armazenar as informações de anúncios e de usuários.

5.1 CASOS DE USO

A partir dos requisitos funcionais listados, foram criados casos de uso, os quais descrevem as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários. Foram descritas as principais funcionalidades do sistema, do ponto de vista do usuário cadastrado e do administrador do sistema.

Na figura 3 estão ilustrados os casos de uso de um usuário cadastrado. Ele poderá realizar as seguintes atividades: Visualizar a lista de anúncios de aluguel indexados; cadastrar-se na aplicação; receber recomendações de anúncios baseado nas preferências de usuário;

Figura 3 Caso de uso para ator do tipo usuário cadastrado



5.2 TECNOLOGIAS UTILIZADAS NA APLICAÇÃO

Há uma grande variedade de frameworks e bibliotecas para criação de um aplicativo web, mas se optou pelo uso das tecnologias contidas na *stack MERN* (*MongoDB, Express, React e NodeJS*). A razão por trás da escolha dessa pilha é a popularidade do *NodeJS* para criar *APIs Rest* (definidos na seção 3.7), a flexibilidade do *Express* para criação das *APIs*, o facilidade da criação de UIs através do React e finalmente a simplicidade de armazenamento de dados fornecida pelo *MongoDB*. Além disso, a aplicação "*VagouAqui*" emprega uma variedade de outras tecnologias a fim de oferecer uma experiência de usuário

eficiente e escalável. No lado do cliente, o gerenciamento de estado é fornecido pelo *React Redux*. O roteamento é tratado com o *React Router*, o que envolve a gestão das rotas (*URLs*) da aplicação, determinando qual componente deve ser renderizado com base na *URL* atual. Isso permite a navegação entre diferentes páginas ou seções de uma aplicação *web* sem a necessidade de recarregar a página inteira a cada mudança de *URL*. Além disso, as requisições HTTP são tratadas de forma eficiente pela biblioteca Axios, garantindo a comunicação eficaz entre o frontend e o backend da aplicação.

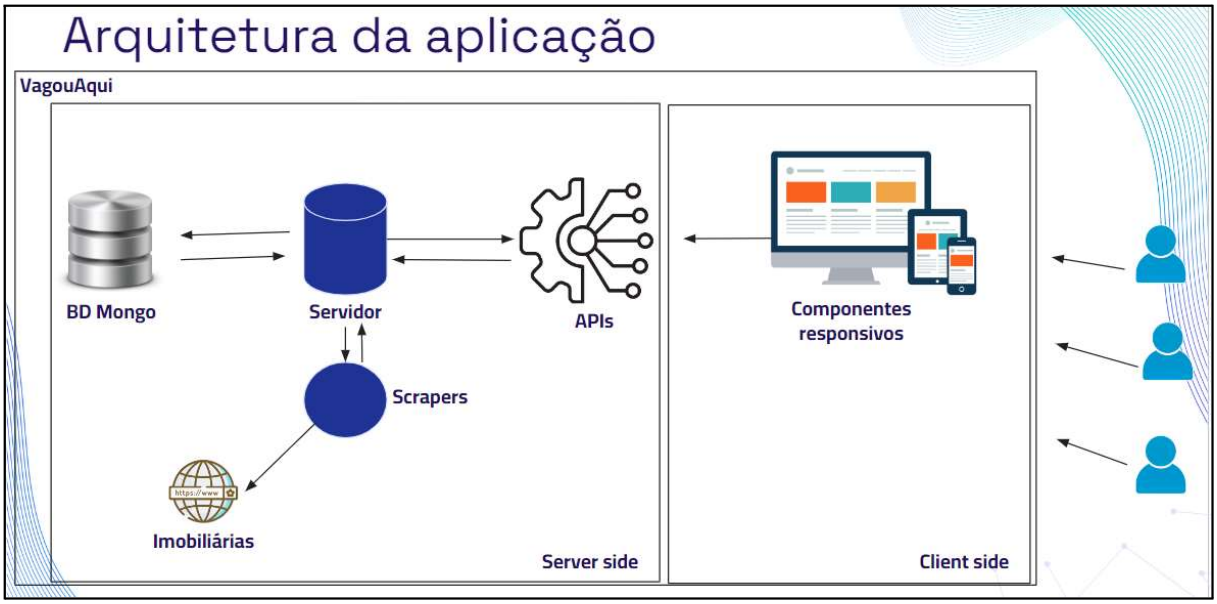
No lado do servidor, a autenticação é implementada com *JSON Web Tokens (JWT)* e senhas são criptografadas usando o *Bcrypt*. A aplicação também inclui recursos de raspagem de conteúdo web com a ajuda da biblioteca Cheerio e agendamento de tarefas com as bibliotecas *Cron* e *Schedule*.

O código-fonte do *VagouAqui* está hospedado na plataforma Codigos@UFSC e pode ser acessado através do link https://codigos.ufsc.br/j.r.junior/VagouAqui4 . O repositório é público sob a licença MIT, a qual é uma licença permissiva curta e simples com condições que exigem apenas a preservação de direitos autorais.

5.3 ARQUITETURA DA APLICAÇÃO

Na Figura 4 há uma ilustração da arquitetura da aplicação. O servidor gerencia os *scrapers,* os quais fazem a coleta de anúncios dos sites de imobiliárias. Os dados coletados e os usuários cadastrados são armazenados dentro do banco de dados Mongodb e eles são servidos para o *front-end* através de *APIs* criadas com o express. O *front-end* é composto de componentes *React*, os quais são utilizados pelos usuários para interagir com a aplicação.

Figura 4 Arquitetura da aplicação VagouAqui



5.4 ESTRUTURA DA APLICAÇÃO

A Figura 5 a seguir apresenta a árvore de diretórios do projeto desenvolvido e os arquivos de cada diretório da aplicação.

Figura 5 Árvore de arquivo da aplicação *VagouAqui*

```
VagouAqui
|   LICENSE.md
|   README.md
|
+---client
|   |   .gitignore
|   |   package-lock.json
|   |   package.json
|   |
|   +---public
|   |       index.html
|   |
|   \---src
|       |   App.js
|       |   index.js
|       |   store.js
|       |
|       +---actions
|       |       authActions.js
|       |       types.js
|       |
|       +---reducers
|       |       authReducer.js
|       |       errorReducer.js
|       |       index.js
|       |
|       +---utils
|       |       PrivateRoute.js
|       |       setAuthToken.js
|       |
|       \---views
|           +---adGrid
|           |       AdGrid.css
|           |       AdGrid.js
|           |       RecommendedGrid.css
|           |       RecommendedGrid.js
|           |
|           +---layout
|           |       Landing.css
|           |       Landing.js
|           |       MainPage.css
|           |       MainPage.js
|           |       Navbar.css
|           |       Navbar.js
|           |
|           \---User
|                   EditPreferences.css
|                   EditPreferences.js
|                   Login.css
|                   Login.js
|                   Register.css
|                   Register.js
|                   UserPage.css
|                   UserPage.js
|
\---server
    |   .env
    |   .gitignore
    |   package-lock.json
    |   package.json
    |   server.js
    |
    +---config
    |       keys.js
    |       passport.js
    |
    +---controller
    |       adsController.js
    |
    +---models
    |       Ads.js
    |       User.js
    |
    +---recommendations
    |       adRecommendation.js
    |
    +---routes
    |   \---api
    |           ads.js
    |           recommendation.js
    |           users.js
    |
    +---scrapers
    |       classificadosUfscScraper.js
    |       contactInfoScraper.js
    |       ibagyScraper.js
    |       imageScraper.js
    |       mgfScraper.js
    |       vivaRealScraper.js
    |       webQuartoScraper.js
    |
    \---validation
            login.js
            register.js
```

A duração de cada etapa do desenvolvimento está representada na Figura 6.

Figura 6 Representação gráfica de Gantt das etapas de desenvolvimento

Na Tabela 1 abaixo há uma descrição geral de cada arquivo ilustrado anteriormente na Figura 5:

Tabela 1 Descrição da árvore de diretório da aplicação ‘VagouAqui’

Client	
.gitignore	Arquivo que lista os arquivos e diretórios que devem ser ignorados pelo controle de versão Git.
package.json	Arquivo que contém informações sobre as dependências do projeto e scripts para gerenciar o projeto Node.js.
README.md	Arquivo de documentação que fornece informações sobre o projeto.
LICENSE.md	Arquivo que contém a licença utilizada para tornar o código-fonte open source.
public	Diretório para ativos públicos, como o arquivo HTML que serve como ponto de entrada para a aplicação no lado *Client*
index.html	Arquivo HTML principal da aplicação web.
src	Diretório que contém todo o código-fonte *Client* da aplicação.
App.js	Arquivo JavaScript que contém a lógica principal da aplicação.

index.js	Arquivo JavaScript de ponto de entrada da aplicação.
reportWebVitals.js	Arquivo JavaScript para relatar métricas de desempenho da aplicação.
store.js	Arquivo JavaScript que define o armazenamento central (store) da aplicação.
actions	Diretório para criadores de ações Redux.
authActions.js	Arquivo JavaScript com ações relacionadas à autenticação de usuário.
types.js	Arquivo JavaScript com definições de tipos de ação.
reducers	Diretório que contém os Redutores Redux para o cliente
authReducer.js	Redutor de autenticação
errorReducer.js	Redutor de erros
index.js	Arquivo de exportação de redutores
utils	Diretório para scripts auxiliares.
setAuthToken.js	Função para configurar token de autenticação
views	Diretório que armazena diferentes visualizações ou páginas em seu aplicativo
adGrid	Diretório que armazena os componentes relacionados à exibição de anúncios e grades recomendadas.
AdGrid.css	Estilos CSS para AdGrid.
AdGrid.js	Componente JavaScript para exibir anúncios.
RecommendedGrid.css	Estilos CSS para RecommendedGrid.
RecommendedGrid.js	Componente JavaScript para exibir recomendações.
layout	Diretório que contém os componentes relacionados ao layout da sua aplicação.
Landing.css	Estilos CSS para a página de aterrissagem.
Landing.js	Componente JavaScript para a página de aterrissagem.
MainPage.css	Estilos CSS para a página principal.
MainPage.js	Componente JavaScript para a página principal.
Navbar.css	Estilos CSS para a barra de navegação.
Navbar.js	Componente JavaScript para a barra de navegação
privateRoute	Diretório que armazena os componentes para lidar com rotas privadas (por exemplo, autenticação).
PrivateRoute.js	Componente JavaScript para rota privada.

User	Diretório com componentes relacionados ao gerenciamento de usuários e páginas de usuários.
EditPreferences.css	Estilos CSS para edição de preferências.
EditPreferences.js	Componente JavaScript para edição de preferências.
Login.css	Estilos CSS para a página de login.
Login.js	Componente JavaScript para a página de login.
Register.css	Estilos CSS para a página de registro.
Register.js	Componente JavaScript para a página de registro.
UserPage.css	Estilos CSS para a página do usuário.
UserPage.js	Componente JavaScript para a página do usuário.
Server	
.env	Arquivo de configuração de ambiente
.gitignore	Arquivo de configuração do Git
package.json	Arquivo de configuração do pacote do servidor
server.js	O principal ponto de entrada para a aplicação no *Server-sider*.
config	Diretório que contém os arquivos de configuração do servidor, como chaves de API e configuração do Passport.js
keys.js	Chaves de configuração
passport.js	Configurações do Passport
controller	Diretório que contém os Controladores do servidor que lidam com a lógica de negócio da aplicação.
adsController.js	Controlador para anúncios
models	Diretório que contém os modelos de dados (Schemas) do servidor.
Ads.js	Modelo de dados para anúncios
User.js	Modelo de dados para usuários
recommendations	Diretório que armazena os arquivos de recomendações de anúncios.
adRecommendation.js	Lógica de recomendação de anúncios
routes	Diretório que contém as rotas de API, incluindo anúncios, recomendações e rotas de usuários.
api	Diretório das rotas da API
ads.js	Rota para anúncios.

recommendation.js	Rota para recomendações.
users.js	Rota para usuários.
scrapers	Diretório que contém os scripts de raspagem para extrair dados de anúncios de diferentes fontes.
classificadosUfscScraper.js	Raspador para dados do Classificados UFSC.
contactInfoScraper.js	Raspador auxiliar para extração de informações de contato.
ibagyScraper.js	Raspador para dados de anúncio da plataforma Ibagy.
imageScraper.js	Raspador auxiliar para extração de dados de imagens.
mgfScraper.js	Raspador para dados de anúncio da plataforma MGF.
vivaRealScraper.js	Raspador para dados de anúncio da plataforma VivaReal.
webQuartoScraper.js	Raspador para dados de anúncio da plataforma WebQuarto.
validation	Diretório que contém as validações de dados
login.js	Validação de login
register.js	Validação de registro

5.5 MÓDULOS EMPREGADOS NA APLICAÇÃO

Além da *stack MERN*, foram utilizadas outras bibliotecas na aplicação. No lado do cliente, são utilizados o *React*, *Redux* para gerenciamento de estado, e várias outras para funcionalidades específicas, como roteamento com react-router-dom e manipulação de imagens com *react-image-lightbox*. O servidor, por sua vez, utiliza *Node.js* com *Express* para a criação de uma *API*, empregando bibliotecas como *Mongoose* para interação com um banco de dados *MongoDB*, e *Passport* para autenticação via *JWT*. Há também recursos de segurança, como *bcryptjs* para a criptografia de senhas, além de ferramentas de validação de dados como validator. Na Tabela 2 há a descrição detalhada de todas as tecnologias empregadas.

Tabela 2 Descrição dos pacotes Node utilizados na aplicação ‘VagouAqui’

Client	
classnames (Versão 2.2.6)	Uma biblioteca para gerar nomes de classes CSS condicionalmente no React.

is-empty (Versão 1.2.0)	O mesmo pacote que no servidor, usado no cliente para verificar se objetos ou strings estão vazios.
jwt-decode (Versão 2.2.0)	Uma biblioteca para decodificar tokens JWT no cliente, frequentemente usada para obter informações do usuário autenticado.
prop-types (Versão 15.8.1)	Uma biblioteca para declarar tipos esperados de propriedades em componentes React.
react (Versão 16.14.0)	A biblioteca principal para desenvolvimento de interfaces do usuário no cliente.
react-dom (Versão 16.14.0)	O pacote para renderização do React no navegador.
react-image-lightbox (Versão 5.1.4)	Uma biblioteca para exibir imagens em uma caixa de luz (lightbox) no React.
react-redux (Versão 5.1.1)	Uma biblioteca que facilita o gerenciamento do estado da aplicação no React, especialmente quando combinado com o Redux.
react-responsive-carousel (Versão 3.2.23)	Uma biblioteca para criar carrosséis responsivos no React.
react-router-dom (Versão 4.3.1)	Uma biblioteca para roteamento de páginas no React, útil para criar aplicativos de várias páginas.
react-scripts (Versão 2.1.1)	Uma coleção de scripts para criar e gerenciar aplicativos criados com o Create React App.
react-scripts (Versão 2.1.1)	Uma biblioteca para criar componentes de seleção personalizados no React.
redux (Versão 4.0.1)	Uma biblioteca para gerenciamento de estado no cliente, frequentemente usada em conjunto com o React.
redux-thunk (Versão 2.3.0)	Um middleware para o Redux que permite lidar com ações assíncronas.
web-vitals (Versão 3.4.0)	Uma biblioteca para medir e rastrear métricas de desempenho na web.
Server	
axios (Versão 1.4.0)	Uma biblioteca para fazer requisições HTTP a partir do servidor, útil para se comunicar com outros serviços ou APIs.
axios-rate-limit (Versão 1.3.0)	Um módulo que permite limitar a taxa de requisições feitas com o Axios, ajudando a evitar sobrecarga no servidor.
axios-retry (Versão 3.8.0)	Adiciona funcionalidade de repetição automática para requisições Axios, útil para lidar com possíveis falhas de rede.
bcryptjs (Versão 2.4.3)	Uma biblioteca para criptografia de senhas, comumente usada para armazenar senhas com segurança no banco de dados.
body-parser (Versão 1.20.2)	Um middleware para o Express que ajuda a analisar o corpo das requisições HTTP, permitindo o acesso aos dados enviados no corpo

	das requisições.
cheerio (Versão 1.0.0-rc.12)	Uma biblioteca para analisar e manipular documentos HTML, frequentemente usada para fazer web scraping ou análise de páginas da web.
concurrently (Versão 8.2.0)	Uma ferramenta para executar várias tarefas em paralelo, o que é útil para iniciar o servidor e o cliente simultaneamente durante o desenvolvimento.
cors (Versão 2.8.5)	Um middleware do Express que lida com políticas de segurança do navegador, permitindo ou bloqueando solicitações de origens diferentes (Cross-Origin Resource Sharing).
dotenv (Versão 16.3.1)	Uma biblioteca para carregar variáveis de ambiente a partir de um arquivo .env, o que é útil para armazenar configurações sensíveis.
express (Versão 4.18.2)	Um framework web para Node.js, frequentemente utilizado para criar APIs e servidores web.
is-empty (Versão 1.2.0)	Um pacote simples para verificar se um objeto ou string está vazio.
jsonwebtoken (Versão 9.0.1)	Uma biblioteca para criar e verificar tokens de autenticação JWT (JSON Web Tokens), frequentemente utilizada em sistemas de autenticação.
lodash (Versão 4.17.21)	Uma biblioteca JavaScript que fornece funções de utilidade para manipulação de dados.
mongodb (Versão 5.7.0	O driver oficial do MongoDB para Node.js, usado para se conectar e interagir com bancos de dados MongoDB.
mongoose (Versão 7.4.2)	Uma biblioteca para modelagem de objetos MongoDB, que facilita a definição de esquemas e a interação com bancos de dados MongoDB.
node-cron (Versão 3.0.2)	Um agendador de tarefas para Node.js que permite executar funções em horários programados.
node-fetch (Versão 3.3.2)	Um módulo para fazer requisições HTTP semelhante ao fetch do navegador, mas no ambiente Node.js.
node-schedule (Versão 2.1.1)	Uma biblioteca para agendar tarefas em horários específicos no Node.js.
passport (Versão 0.6.0)	Um middleware de autenticação para Express, frequentemente usado para autenticar usuários em aplicativos da web.
passport-jwt (Versão 4.0.1)	Uma extensão do Passport que permite autenticar usuários usando tokens JWT.
validator (Versão 13.11.0)	Uma biblioteca para validação de dados, incluindo validação de strings, números e outros tipos de entrada.
moment-timezone (Versão 0.5.43)	Uma biblioteca JavaScript que estende a funcionalidade da popular biblioteca "moment.js" adicionando suporte a fuso horário e "moment.js" é uma biblioteca para analisar, validar, manipular e formatar datas e horas em JavaScript.

5.6 DESENVOLVIMENTO DO *SCRAPERS* UTILIZADOS NA APLICAÇÃO

Para a aplicação foram desenvolvidos 5 raspadores *(scrapers)* de dados de sites de aluguéis em Florianópolis. As plataformas escolhidas e o respectivo *scraper* utilizado são:

- Classificados UFSC - classificadosUfscScraper.js
- Ibagy Imóveis - ibagyScraper.js
- MGF Imóveis - mgfScraper.js
- VivaReal Imóveis - vivaRealScraper.js
- Web Quarto Imóveis - webQuartoScraper.js

Além disso, 2 raspadores auxiliares foram desenvolvidos para dar assistência no processo de raspagem de dados. São eles:

- ImageScraper.
- ContactInfoScraper.

Apesar de cada *scraper* possuir características e lógica de negócio distintas, todos eles possuem as 3 etapas do processo de *web scraping* descrito na seção 3.7.1 deste documento. Nas próximas Seções há mais detalhes acerca da lógica empregada no controlador de *scrapers*, bem como a descrição detalhada do funcionamento de cada *scraper*.

5.6.1. CONTROLADOR SCRAPER

O arquivo *adsController.js* é o controlador de todos os *scrapers* utilizados na aplicação. Ele é responsável tanto pela chamada de cada um dos raspadores, como também é responsável por salvar os anúncios que são retornados das chamadas. Para tanto, há a função *startScraping*, a qual orquestra todo o processo de *scraping*. Ela chama a função *scrapeAndSaveNewAds* para várias fontes de anúncios, como "Classificados UFSC", "Ibagy", "WebQuarto", "vivaReal" e "MGF Imóveis". Ela também atualiza uma variável de ambiente chamada *LAST_SCRAPING_DATE* com a data e hora da última execução bem-sucedida do processo de scraping. Por ser uma variável de ambiente, pode ser facilmente acessada em toda a aplicação, fornecendo a informação temporal crucial sobre a última execução bem-sucedida do scraping. Além disso, seu caráter ambiental permite que essa informação seja

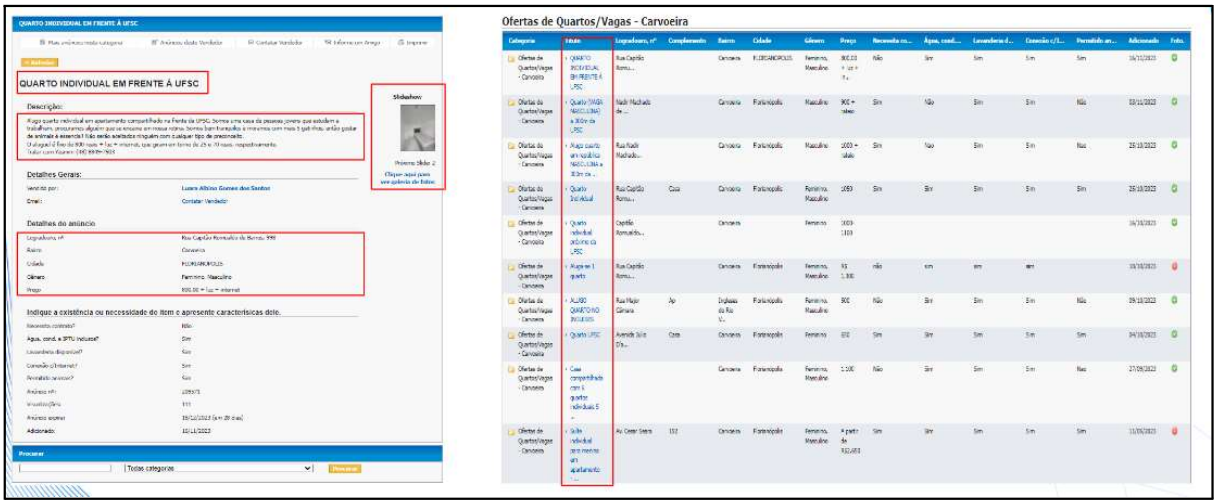
mantida e utilizada entre diferentes partes do código, simplificando a gestão do tempo de execução do scraping.

A função *scrapeAndSaveNewAds* é responsável por realizar o *scraping* e salvar novos anúncios. Ela recebe parâmetros que incluem o scraper a ser usado, a fonte dos anúncios, as URLs a serem rastreadas, a função de scraping e a função para obter detalhes dos anúncios. O código começa verificando se há URLs fornecidas. Se houver, ele itera sobre cada URL. Para cada URL, ele chama a função de scraping especificada para coletar os anúncios dessa fonte. Em seguida, ele busca os anúncios já existentes no banco de dados. Os novos anúncios são filtrados, ou seja, apenas aqueles que não existem no banco de dados são mantidos. Se novos anúncios forem encontrados, ele pode chamar a função de obtenção de detalhes, se essa função estiver definida, para obter mais informações sobre os anúncios. Os novos anúncios são mapeados e formatados no formato *JSON* e, em seguida, salvos no banco de dados utilizando a função *saveNewAds*.

5.6.2 SCRAPER CLASSIFICADOSUFSCSCRAPER.JS

Este *scraper* tem a tarefa de coletar informações de anúncios dos Classificados UFSC, o qual está hospedado no endereço "classificados.inf.ufsc.br". Primeiramente, ele importa as bibliotecas necessárias, como Axios para fazer solicitações HTTP, Cheerio para analisar e extrair dados HTML, e outras bibliotecas para controlar a taxa de solicitações e retentativas. Em seguida, ele configura uma instância do Axios chamada *axiosInstance*. Essa instância é importante porque controla o número de solicitações que podemos fazer por segundo (limitado a 2) e também permite retentar uma solicitação até 3 vezes se algo der errado. Feito isso, a função *getAdLinks* entra em cena: ela recebe uma URL como entrada e realiza uma solicitação *HTTP* para essa URL. Depois, ela analisa a página HTML resultante usando o Cheerio e extrai os links dos anúncios válidos. Esses links são armazenados em uma lista chamada *items* e, por fim, são retornados. Na Figura 7 há exemplos da página principal que contém os links dos anúncios e os elementos de um anúncio individual que são extraídos da página.

Figura 7. Elementos que são coletados pelo *scraper* CLASSIFICADOSUFSC.JS



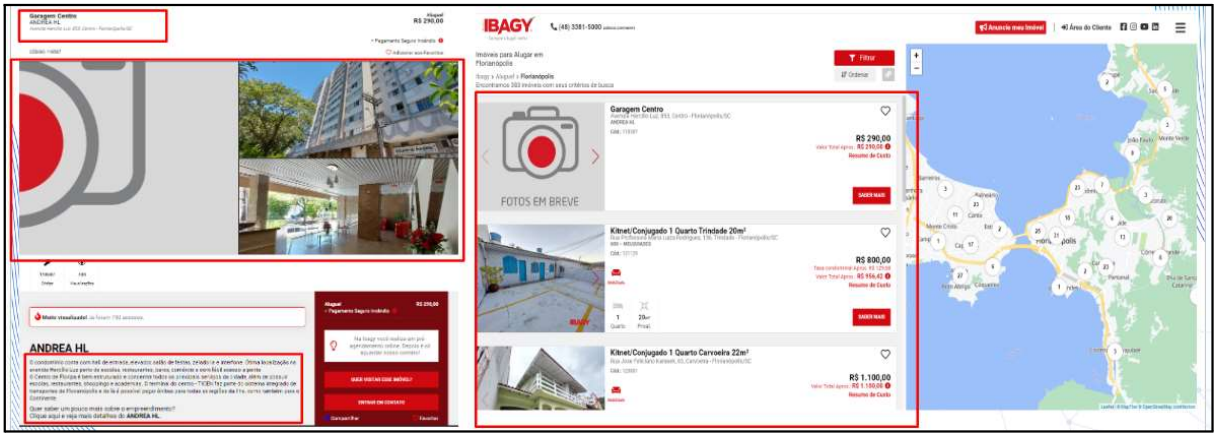
Existe também uma função auxiliar chamada *isValidLink*. Ela desempenha um papel crucial verificando se um link corresponde a um padrão específico de URL do site de classificados. Isso ajuda a filtrar apenas os links válidos antes de salvá-los no banco de dados. A função *getAdDetails* é responsável por pegar a lista de links de anúncios (que foram obtidos anteriormente) e, para cada anúncio, fazer uma solicitação HTTP adicional para obter detalhes específicos sobre esse anúncio. Isso inclui informações como preço, gênero, bairro, título, descrição, imagens e informações de contato. Dentro dessa função, o código utiliza o Cheerio para analisar o HTML da página de detalhes de cada anúncio e extrair essas informações. Também lida com casos em que as informações estão ausentes ou formatadas de maneira incomum. Quando as informações de um anúncio são coletadas com sucesso, elas são armazenadas em objetos e adicionadas a uma lista chamada *itemsWithDetails*. Se ocorrer algum erro durante esse processo, o código registra esses erros no console para futura análise. Após processar todos os anúncios, a função *getAdDetails* retorna a lista *itemsWithDetails*, que contém os detalhes de todos os anúncios válidos.

5.6.3 SCRAPER IBAGYSCRAPER.JS

Este *scraper* realiza raspagem de dados de anúncios imobiliários no site "ibagy.com.br". O objetivo principal é coletar informações como título, descrição, preço, endereço, bairro, informações de contato e links para imagens relacionadas a esses anúncios.

O script começa importando as bibliotecas necessárias, como o *Axios* para fazer requisições *HTTP*, o *Axios Rate Limit* para limitar a taxa de solicitações, o *Cheerio* para analisar o *HTML* das páginas da web e duas funções personalizadas, "*scrapeImagesIbagy*" e "*extractIdfromAdLink*," que ajudam a obter informações adicionais e serão discutidas quando os *scrapers* auxiliares forem abordados. Em seguida, ele define uma função assíncrona chamada "*scrapeIbagyAds*." Esta função é responsável por realizar a raspagem das páginas do site "ibagy.com.br" que contém links para anúncios de imóveis para aluguel em Florianópolis. Ele define uma matriz de *URLs* das páginas e, em seguida, usa um mapeamento assíncrono para buscar essas páginas em paralelo. A biblioteca *Axios Rate Limit* é usada para garantir que não sejam feitas mais de 2 solicitações por segundo (ajustável com base nas políticas de limitação do site). Para cada página, o script verifica se a resposta *HTTP* é bem-sucedida (código de status 200). Se for, ele usa o *Cheerio* para analisar o *HTML* da página em busca de links para anúncios imobiliários. Os links únicos são coletados e retornados em um array. Se a solicitação não for bem-sucedida, o erro é tratado e uma matriz vazia é retornada. Depois de coletar todos os links para os anúncios, a função *scrapeIbagyAdsDetails* é chamada para coletar detalhes específicos de cada anúncio. Essa função também é assíncrona e lida com a coleta de informações detalhadas de cada anúncio, como título, descrição, preço, endereço, bairro, informações de contato e links para imagens relacionadas. Além disso, a função "*extractNeighborhood*" é usada para extrair o nome do bairro do endereço. A Figura 8 ilustra os elementos coletados pelo *scraper.*

Figura 8. Elementos que são coletados pelo *scraper* IBAGYSCRAPER.JS



Finalmente, as informações detalhadas de cada anúncio são organizadas em objetos JSON e armazenadas em um array chamado *adDetailsArray*. Após a conclusão de todas as solicitações para os detalhes dos anúncios, esse array é retornado.

5.6.4 SCRAPER MGFSCRAPER.JS

O *scraper* utiliza a biblioteca Axios para fazer requisições *HTTP* e a biblioteca Cheerio para analisar e manipular o *HTML* das páginas da web. Além disso, há uma função auxiliar importada chamada *extractContactInfoFromDescription* para extrair informações de contato a partir da descrição dos anúncios. A função principal do scraper é *extractMgfHrefValues*, a qual é assíncrona (async). Essa função tem como objetivo coletar *URLs* de anúncios do site MGF Imóveis. Para isso, ela realiza as seguintes etapas:

- Define a *URL* base do site e um conjunto (Set) vazio chamado *adLinks* para armazenar as *URLs* únicas dos anúncios.

- Cria uma função assíncrona chamada *getPageLinks* para buscar links de páginas diferentes (neste caso, 30 páginas) e extrair as *URLs* dos anúncios de cada página. Cada página é acessada usando *axios.get* e analisada com o *Cheerio*.

- As *Promises* geradas por cada chamada a *getPageLinks* são armazenadas em um array chamado *pagePromises*.

- A função Promise.all é usada para aguardar que todas as *Promises* em *pagePromises* sejam resolvidas antes de continuar com a execução do código.

- Após a conclusão bem-sucedida, as *URLs* coletadas são convertidas de um *Set* para um array e, em seguida, a função *extractMgfAdDetails* é chamada para extrair informações detalhadas dos anúncios.

A função *extractMgfAdDetails* recebe uma lista de *URLs* de anúncios e realiza a raspagem de informações detalhadas de cada anúncio. O processo de scraping inclui a obtenção do título, descrição, preço, bairro, links de imagens e informações de contato. Para coletar as informações detalhadas de cada anúncio, a função cria *Promises* para cada *URL* de anúncio. Cada *Promise* realiza o seguinte processo:

- Faz uma requisição *HTTP* com *axios.get* para a *URL* do anúncio.

- Analisa o *HTML* da página com Cheerio e extrai as informações relevantes, como título, descrição, preço, bairro, imagens e informações de contato.

- As informações extraídas são então usadas para criar um objeto *adDetail* que contém esses dados.

- A Promise retorna o objeto *adDetail* se a requisição e análise forem bem-sucedidas.

- Os erros durante a raspagem são tratados e registrados no console.

Após a conclusão de todas as *Promises*, a função *Promise.all* é usada novamente para aguardar a resolução de todas as *Promises* geradas para os anúncios. O resultado é uma lista de objetos *adDetail* representando informações detalhadas de anúncios. A Figura 9 ilustra o que é extraído pelo scraper.

Figura 9. Elementos que são coletados pelo *scraper* MGFSCRAPER.JS

As *Promises* são usadas nesse processo para lidar com operações assíncronas, como as requisições *HTTP* para várias páginas e anúncios. Isso permite que o código seja executado de forma eficiente, aguardando o término de todas as operações antes de prosseguir para a próxima etapa. Qualquer erro que ocorra durante o processo de scraping é tratado para evitar interrupções na execução do programa. Finalmente, as informações detalhadas de cada anúncio são organizadas em objetos *JSON* e retornado para o controlador.

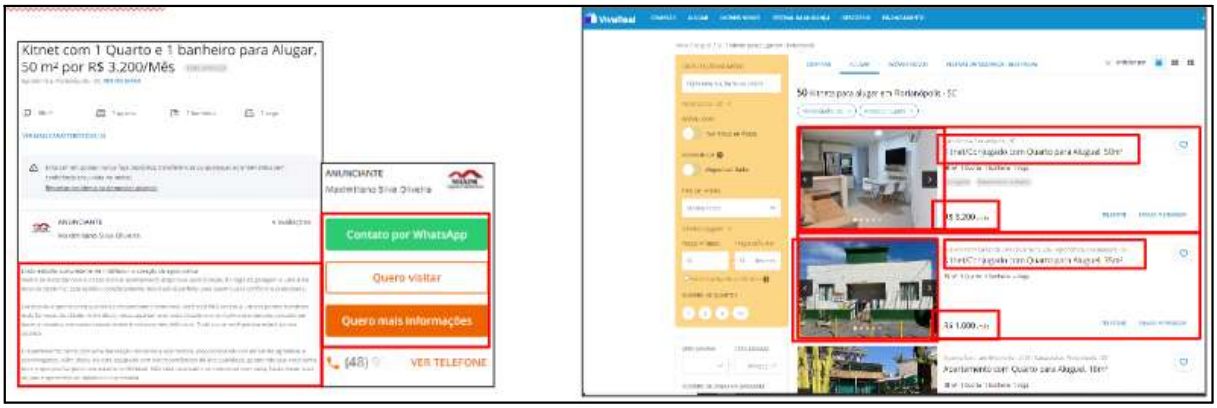
5.6.5 SCRAPER VIVAREALSCRAPER.JS

Este *scraper* foi desenvolvido para coletar informações de anúncios imobiliários no site "VivaReal". Ele utiliza diversas bibliotecas, incluindo Axios para fazer requisições *HTTP*, Cheerio para analisar e extrair dados de páginas *HTML* e a biblioteca "axios-rate-limit" para limitar a taxa de requisições a uma por segundo.

O ponto de partida do script é a função *getVivaRealAdLinks*. Ela começa fazendo uma requisição *HTTP* para a página no site do VivaReal. Após obter o *HTML* da página, utiliza o Cheerio para analisar a estrutura do documento e extrair detalhes de cada anúncio presente.

Dentro do loop que itera sobre os anúncios na página, o script coleta informações como título, link para o anúncio, preço e endereço do imóvel. Neste ponto, ele utiliza a capacidade do JavaScript de trabalhar com *Promises* para gerenciar chamadas assíncronas. O script faz uma chamada para a função *extractDescription* para obter a descrição do anúncio. A função *extractDescription* realiza uma nova requisição *HTTP* para a página de descrição do anúncio e, após obter o *HTML* correspondente, extrai a descrição do imóvel. A utilização de *Promises* permite que o script aguarde a conclusão dessa operação assíncrona antes de prosseguir. Além disso, o script utiliza a função *extractContactInfoFromDescription* para extrair informações de contato da descrição e a função *extractNeighborhood* para tentar extrair o bairro a partir do endereço do imóvel. O código também busca as imagens associadas a cada anúncio, chamando a função *extractVivaRealImageLinks*, que faz uma nova requisição *HTTP* para a página do anúncio e extrai os links das imagens. Novamente, a utilização de *Promises* é essencial para garantir que todas as operações de scraping sejam concluídas antes de continuar. A Figura 10 a seguir ilustra os elementos coletados.

Figura 10. Elementos que são coletados pelo *scraper* VIVAREALSCRAPER.JS



Todas as informações coletadas, como título, descrição, preço, endereço, informações de contato, bairro e links das imagens, são agregadas em objetos e posteriormente armazenadas em um array de objetos. Ao final do processo, a função *getVivaRealAdLinks* retorna esse array de objetos *JSON* contendo todas as informações coletadas dos anúncios.

5.6.6 SCRAPER WEBQUARTOSCRAPER.JS

Este *scraper* realiza a raspagem de dados de anúncios de quartos no site "webquarto.com.br". O script é dividido em várias partes que compõem a lógica geral do processo de raspagem, as quais são:

Importação de bibliotecas e módulos: O código começa importando as bibliotecas e módulos necessários para a raspagem de dados, como o Axios (para fazer solicitações HTTP), Cheerio (para fazer o parsing do HTML), um módulo personalizado chamado *contactInfoScraper*, e a*xios-rate-limit* para limitar a taxa de solicitações HTTP.

Configuração do Axios com limite de taxa: Em seguida, é criada uma instância do Axios que é limitada a fazer no máximo 1 solicitação por segundo. Isso é feito para evitar sobrecarregar o servidor alvo com muitas solicitações em um curto período de tempo.

Função para raspar uma única página: A função *scrapeWebQuartoadsPage* é definida para raspar os dados de uma página específica. Essa função recebe um número de página como argumento e realiza as seguintes etapas:

- Construir a URL da página com base no número da página.

- Fazer uma solicitação HTTP para a URL usando a instância limitada do Axios.

- Verificar se a resposta HTTP tem status 200 (sucesso).

- Usar a biblioteca Cheerio para analisar o HTML da página.

- Extrair os dados relevantes do HTML, que incluem informações sobre anúncios de quartos.

- Organizar os dados em um formato estruturado e retornar a lista de anúncios. Se a solicitação falhar ou ocorrer um erro durante o processo, ele é tratado e uma mensagem de erro é exibida no console.

Função para raspar todas as páginas: A função *scrapeWebQuartoads* é definida para raspar todos os anúncios da página 1 a 8 e cada página possui 16 anúncios. Essa função faz a iteração por todas as páginas e chama a função *scrapeWebQuartoadsPage* para cada página. Na figura 11 abaixo é possível ver a página principal e o JSON que contém todas as informações de anúncios que são coletadas pelo *scraper*.

Figura 11. Elementos que são coletados pelo *scraper* WEBQUARTOSCRAPER.JS

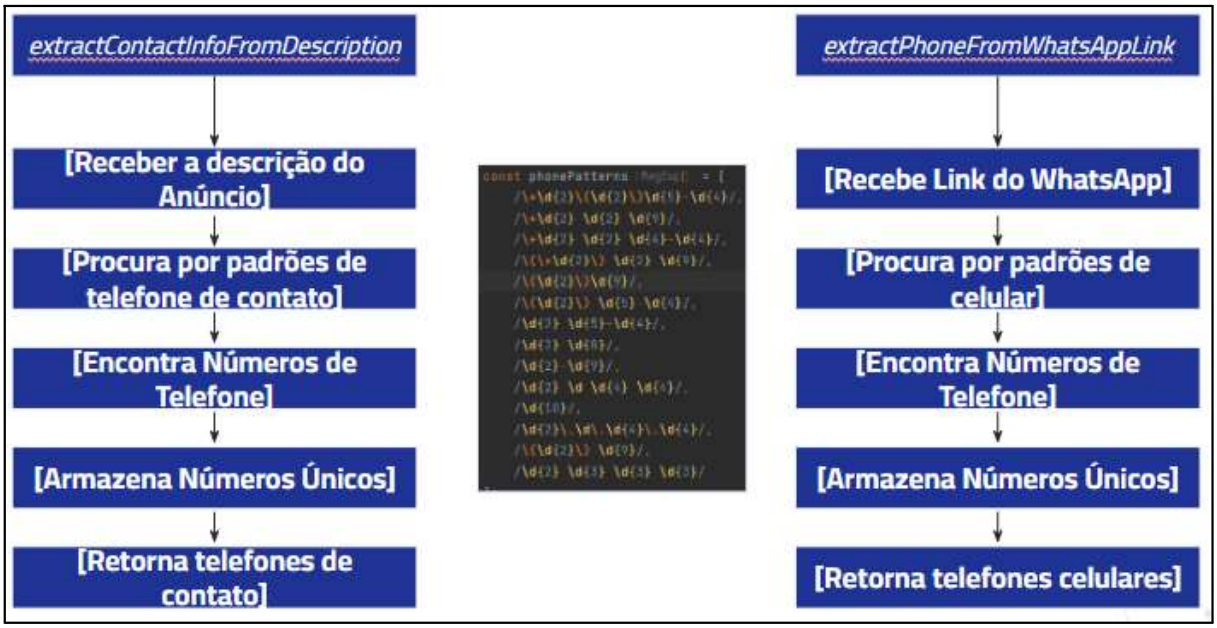
Os dados de anúncios obtidos em cada página são agregados em uma única lista chamada *allAds* e essa lista é retornada quando todas as páginas são processadas. Por fim, a função *scrapeWebQuartoads* é exportada para que possa ser usada em outros lugares.

5.6.7 SCRAPER AUXILIAR CONTACTINFOSCRAPER.JS

Este *scraper* contém três funções auxiliares distintas, cada uma projetada para extrair informações específicas de contato de diferentes tipos de entrada. A primeira função, chamada *extractContactInfoFromDescription*, recebe uma descrição como entrada, que é uma string. Ela opera com um conjunto de expressões regulares (*regex*) armazenadas em um array chamado *phonePatterns*, destinadas a identificar números de telefone em diversos formatos. A função, então, cria um conjunto *(Set)* chamado *uniqueContactInfo* para armazenar números de telefone únicos encontrados na descrição. Iterando através das expressões regulares em *phonePatterns*, a função utiliza o método .match() para procurar correspondências na descrição. Ao encontrar uma correspondência, ela adiciona o primeiro número de telefone encontrado ao conjunto *uniqueContactInfo*. No final, a função converte o conjunto de números de telefone únicos em um array e retorna esse array como resultado.

A segunda função, denominada *extractPhoneFromWhatsAppLink*, recebe como entrada um link de API do WhatsApp, que é uma string. Nessa função, é definida uma expressão regular chamada regex para extrair o número de telefone do link. A expressão regular procura por um padrão em que o número de telefone é precedido por *?phone*= ou *&phone*=. A função utiliza o método *.match()* com essa expressão regular para buscar uma correspondência no link. Se uma correspondência for encontrada e o primeiro grupo de captura (indicado por *match[1]*) contiver o número de telefone, a função retornará esse número. Por outro lado, se nenhuma correspondência for encontrada ou o número de telefone não estiver presente no grupo de captura, a função retornará null. Na figura 12 há um resumo de como é a rotina das funções e os padrões *Regex* que são utilizados para identificar o telefone de contato.

Figura 12. Funcionamento dos scrapers auxiliares



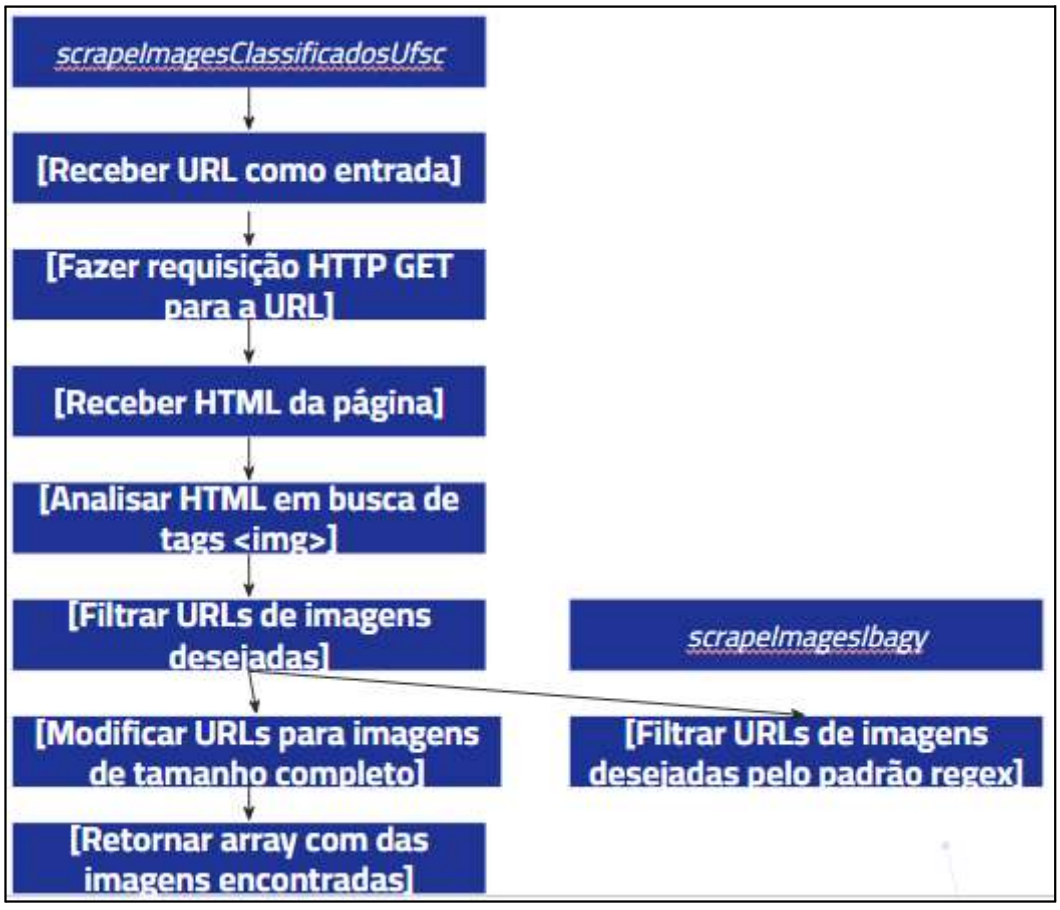
5.6.8 SCRAPER AUXILIAR IMAGESCRAPER.JS

Este *scraper* tem como principal finalidade extrair *URLs* de imagens de duas fontes diferentes. Inicialmente, é definido um padrão de *URL* por meio de uma expressão regular chamada *imageUrlPattern*, que será usada posteriormente para filtrar *URLs* de imagens específicas que contenham o padrão "https://cdn.vistahost.com.br/ibagyimo/vista.imobi/fotos/".

O *scraper* define 2 funções assíncronas: *scrapeImagesClassificadosUfsc* e *scrapeImagesIbagy*. A primeira função recebe um parâmetro *imageUrls*, que provavelmente é uma *URL* de uma página da web, e retorna uma promessa (*Promise*). A função realiza uma solicitação *HTTP* para a *URL* fornecida utilizando *https.get*. À medida que a resposta é recebida, os dados brutos são acumulados na variável *data*. Após a conclusão da coleta de dados, a biblioteca cheerio é usada para carregar o *HTML* da página. Em seguida, a função percorre todas as tags *<img>* no *HTML* e extrai seus atributos src, armazenando os *URLs* das imagens em um array denominado *images*. Posteriormente, a função filtra as *URLs* das imagens de acordo com critérios específicos, criando um novo array chamado *adImages*. Esse array é composto por *URLs* de imagens que contenham a substring *"images" e "_tmb1.jpg"* no caminho, e esses *URLs* são transformados em *URLs* completas que correspondem a um formato específico. O resultado final é resolvido na promessa.

A segunda função, *scrapeImagesIbagy*, é semelhante à primeira, mas é projetada para coletar imagens de uma página da web que correspondam a um padrão definido na expressão regular *imageUrlPattern*. A função recebe uma *URL* como parâmetro e retorna uma promessa que será resolvida com um array de *URLs* únicos de imagens que atendem a esse padrão. A função também utiliza *https.get* para fazer uma solicitação *HTTP* e, em seguida, carrega o *HTML* da página com Cheerio. Após a análise do *HTML*, a função coleta os *URLs* de imagens que correspondem ao padrão definido e os armazena em um conjunto *(Set)* para garantir que apenas *URLs* únicos sejam mantidos. Finalmente, o conjunto de *URLs* é transformado em um array e é resolvido na promessa. Na imagem 13 há um resumo do funcionamento das funções.

Figura 13. Funcionamento dos scrapers auxiliares



5.7 MODELO DE DADOS DE ANÚNCIOS (AD SCHEMA)

O primeiro modelo, denominado *adSchema* foi construído com a biblioteca Mongoose, é essencial para armazenar informações detalhadas sobre os anúncios de imóveis. Nele, são definidos diversos campos essenciais para a descrição de um anúncio. O campo "title" (Título) é uma string que contém o título do anúncio, enquanto o campo "link" (Link) é outra string destinada a armazenar o link correspondente ao anúncio. Além disso, o campo "description" (Descrição) permite uma descrição mais detalhada do imóvel, enquanto "price" (Preço) armazena o valor associado ao anúncio. Outros campos incluem "imageLinks" (Links de Imagem), um array que contém *URLs* de imagens relacionadas ao anúncio, "neighborhood" (Bairro) para especificar a localização do imóvel, "contactInfo" (Informações de Contato), um array que pode conter dados de contato relevantes, e "source" (Fonte) que identifica a origem do anúncio. Através desses campos, o schema de anúncios proporciona uma estrutura organizada para armazenar informações cruciais sobre imóveis disponíveis no sistema.

5.8 MODELO DE DADOS DE USUÁRIOS (USER SCHEMA)

O *schema* de usuário, chamado de *UserSchema*, é fundamental no sistema, pois representa as informações de perfil dos usuários. Este *schema* inclui campos como *"name"* (Nome), *"email"* (E-mail) e *"password"* (Senha), que são informações básicas de identificação e autenticação. O campo *"date"* (Data) registra a data de criação do perfil do usuário, com uma data padrão, o que é importante para auditoria e acompanhamento. Além disso, existe o segundo *schema* conhecido como *PreferencesSchema*, que permite os usuários expressem suas escolhas e requisitos específicos relacionados a imóveis. Para isso, o *schema* inclui vários campos que refletem as preferências dos usuários, como: campo "*houseOrApartment*" (Casa ou Apartamento), os usuários podem escolher entre "casa" ou "apartamento" como sua preferência de moradia. O campo "*genderPreference*" (Preferência de Gênero) permite que os usuários indiquem suas preferências em relação aos colegas de moradia, incluindo opções como "masculino," "feminino," "tanto faz," e "ambos." O campo "*acceptsPets*" (Aceita Animais de Estimação) é um booleano que indica se o usuário está disposto a permitir animais de estimação em seu imóvel.

As preferências dos usuários, capturadas pelo *PreferencesSchema*, são incorporadas ao *UserSchema* como parte das informações de perfil do usuário, fornecendo um meio eficaz de capturar as necessidades e desejos dos usuários em relação a sua futura moradia. Isso permite uma experiência personalizada, auxiliando os usuários na busca de imóveis que correspondam às suas preferências e requisitos específicos.

5.9 ROTAS PARA ANÚNCIOS

O arquivo *ads.js* foi criado para armazenar a definição e configuração das rotas HTTP que lidam com os anúncios de aluguel. As rotas são utilizadas para mapear URLs específicas para funções no controlador de anúncios que lidam com as solicitações HTTP correspondentes e o código define 2 delas.

A primeira rota utiliza o método HTTP 'GET' e corresponde ao caminho *'/all'* e, quando uma solicitação GET é feita para essa rota, o controlador '*adController.getAllAds*' é invocado para processar a solicitação e retornar informações sobre todos os anúncios na aplicação.

A segunda rota também utiliza o método 'GET' através do caminho *'/lastScrapingDate'*. Quando uma solicitação GET é feita para essa rota, o controlador '*adController.getLastScrapingDate*' é chamado para buscar e retornar a data da última raspagem de anúncios, a qual será relevante do lado *Client* da aplicação.

5.10 ROTAS PARA USUÁRIOS

O arquivo *users.js* gerencia as rotas do usuário. Elas permitem que os usuários realizem operações relacionadas ao registro, login, atualização de preferências, recuperação de preferências, recuperação de informações do usuário e exclusão de conta, todas acessíveis por meio de uma API RESTful.

A rota */api/users/register* é responsável pelo registro de novos usuários. Inicialmente, o código valida os dados de entrada do usuário, incluindo nome, email, senha e preferências. Se os dados forem válidos, ele verifica se o email já está em uso no banco de dados. Se o email já estiver em uso, ele retorna um erro. Caso contrário, ele cria um novo usuário, criptografa a senha usando o bcrypt, e salva as

informações do usuário no banco de dados. Após um registro bem-sucedido, os dados do usuário são retornados em formato JSON.

A segunda rota, */api/users/login*, lida com o processo de autenticação de um usuário. Ela começa validando os dados de entrada, como email e senha, e verifica a existência do usuário no banco de dados. Se o usuário não for encontrado, ela retorna um erro. Caso contrário, ela verifica se a senha fornecida corresponde à senha armazenada no banco de dados usando o *bcrypt*. Se a senha estiver correta, eacria um *token JWT* contendo informações do usuário, como *ID* e nome, e o retorna como parte da resposta.

A rota seguinte, */api/users/preferences,* permite que um usuário autenticado atualize as preferências dele. Para acessá-la, o usuário deve estar autenticado por meio de um *token JWT*. Quando um usuário envia uma solicitação PUT contendo as preferências atualizadas, o código verifica a autenticação e atualiza as preferências no banco de dados, retornando as preferências atualizadas. Já a rota */api/users/preferences* permite que um usuário autenticado recupere suas preferências. Da mesma forma que as anteriores, ela requer autenticação por meio de um token JWT. O código busca as preferências do usuário no banco de dados e as retorna como parte da resposta.

Por último, a rota */api/users/me* permite que o usuário autenticado recupere suas informações pessoais. Assim como as rotas de preferências, ela requer autenticação por meio de um token JWT. O código busca as informações do usuário no banco de dados e as retorna como parte da resposta.

5.11 FUNÇÃO DE RECOMENDAÇÃO

O sistema de recomendação da aplicação *'VagouAqui'* está definido dentro do arquivo *adRecommendation.js*. Nele há uma função assíncrona chamada *generateRecommendations*, cujo propósito é realizar a busca de anúncios de imóveis em um banco de dados e, com base nas preferências do usuário, gerar recomendações. Para compreender melhor o funcionamento do código, é necessário analisar os detalhes de sua implementação.

A função *generateRecommendations* é definida como assíncrona e recebe um parâmetro denominado *userPreferences*, o qual deve conter as preferências do usuário relacionadas aos anúncios de imóveis. No interior da função, ocorre uma

tentativa de buscar todos os anúncios de imóveis no banco de dados por meio do método *Ad.find({})*. Os anúncios recuperados são armazenados na variável *ads.* Posteriormente, o código estabelece uma variável chamada *sampleSize* para determinar o tamanho da amostra de anúncios que será utilizada para gerar recomendações. Inicialmente é definido um valor padrão, mas esse valor pode ser ajustado com base na quantidade total de anúncios encontrados, pois dependendo do número total de anúncios, representado pela variável *totalAds*, o valor de *sampleSize* é adaptado mediante a aplicação de condições if-else. Caso haja um número muito reduzido de anúncios (menos de 50), todos os anúncios disponíveis serão incluídos na amostra. Se o número de anúncios estiver na faixa de 50 a 500, a amostra terá um tamanho de 100 anúncios. Para quantidades entre 500 e 1000 anúncios, a amostra será mais expressiva, contendo 250 anúncios. Se, por fim, houver mais de 1000 anúncios, a amostra será composta por 300 anúncios. Neste ponto, é criado um array vazio denominado *recommendations*, destinado a armazenar as recomendações de anúncios.

O código segue com a extração da preferência de orçamento do usuário, convertendo-a em um valor numérico a partir do objeto *userPreferences.budget*. Em seguida, inicia-se um loop que percorre cada anúncio presente no array *ads*. Para cada anúncio, se tenta extrair informações relacionadas ao orçamento presentes na descrição do anúncio e as converte em valores numéricos. Se a extração do orçamento não for bem-sucedida ou se o orçamento do anúncio for igual ou inferior ao orçamento do usuário, o código prossegue para o próximo anúncio. Adicionalmente, ele extrai outras características dos anúncios, como o tipo de imóvel, preferência de gênero, aceitação de animais de estimação, localização, disponibilidade de companheiros de quarto, duração do aluguel, acessibilidade para cadeirantes, nível de ruído, aceitação de fumantes e disponibilidade de mobília.

Para cada anúncio, é calculada uma pontuação com base nas preferências do usuário. Cada característica que corresponde às preferências do usuário e à descrição do anúncio contribui com 1 ponto à pontuação. Os anúncios e suas respectivas pontuações são armazenados no array *recommendations*. Após a conclusão do *loop*, as recomendações são ordenadas em ordem decrescente de pontuação, garantindo que os anúncios com as pontuações mais elevadas sejam exibidos no topo da lista.

Por fim, a função retorna as recomendações como um objeto *JSON*, limitando o número de recomendações ao valor previamente definido em *sampleSize*. Caso ocorra algum erro durante a execução, o código captura essa exceção, trata-a e retorna o array vazio como resultado.

5.12 ARQUIVOS AUXILIARES

O servidor também possui uma série de arquivos auxiliares que são componentes essenciais para a autenticação de usuários na aplicação 'VagouAqui', permitindo o registro, login e autenticação por meio de tokens JWT, com verificações detalhadas para garantir a integridade e segurança das operações. São eles *keys.js, passport.js, login.js* e *register.js*, e os detalhes de cada arquivo serão abordados nas próximas seções.

5.12.1 ARQUIVOS *KEYS.JS* E *PASSPORT.JS*

Esses arquivos fazem parte do serviço de autenticação e validação de usuários que utilizam o *framework Passport* e *JSON Web Tokens (JWT)* para proteger rotas e endpoints. Primeiramente no arquivo *keys.js*, é exportado um objeto contendo uma chave chamada *secretOrKey*, que contém uma chave secreta, comumente usada para assinar e verificar tokens *JWT*, garantindo a segurança da autenticação. No arquivo *passport.js* o código carrega as dependências necessárias, como passport-jwt, mongoose, o *schema* de usuário, e as configurações de chaves que foram exportadas. O código define um objeto chamado *opts*, o qual servirá para configurar a estratégia de autenticação JWT do *Passport*, além de especificar como o token JWT deve ser extraído da solicitação e qual chave secreta deve ser usada para verificar a validade do token.

Para extrair o token JWT da solicitação, o código utiliza o método *ExtractJwt.fromAuthHeaderAsBearerToken*, indicando que o token deve ser procurado no cabeçalho de autenticação da solicitação HTTP, seguindo o esquema *Bearer*. A chave secreta que foi definida anteriormente é atribuída a *opts.secretOrKeys*, pois isso permite que se saiba qual chave usar para verificar a autenticidade do token JWT.

O arquivo *passport.js* também possui um módulo exportável contendo uma função que recebe o objeto *passport* como argumento e cria uma instância de *JwtStrategy* e configurada com as opções *opts* definidas anteriormente. Essa classe é projetada para verificar a validade do *token JWT* recebido na solicitação. Para fazer isso, ela utiliza o ID contido no payload do token para procurar um usuário correspondente no banco de dados, utilizando o modelo User do Mongoose. Se um usuário correspondente for encontrado, a função de retorno de chamada é chamada com o usuário como argumento, indicando que a autenticação foi bem-sucedida. No entanto, se nenhum usuário correspondente for encontrado, a função de retorno é chamada com *null* como primeiro argumento e *false* como segundo, indicando que a autenticação falhou. Quaisquer erros que possam ocorrer durante o processo de pesquisa no banco de dados são capturados e registrados no console.

5.12.2 ARQUIVOS *LOGIN.JS* E *REGISTER.JS*

Esses arquivos são módulos de validação para a autenticação de usuários na aplicação, onde o primeiro é responsável por validar os dados de login e o segundo por validar os dados de registro. Ambos fazem uso de duas bibliotecas externas: *validator* e *is-empty*.

O arquivo *login.js* define uma função chamada *validateLoginInput* que aceita um objeto *data* como argumento. Este objeto contém informações do usuário, como o email e a senha fornecidos no processo de login. A função inicia a validação verificando se os campos de email e senha estão vazios e, se estiverem, os converte em strings vazias para que as funções de validação possam ser aplicadas. Em seguida, é verificado se o campo de email está vazio e, se estiver, é adicionado um erro à lista de erros indicando que o campo de email é necessário. Além disso, é verificado se o email fornecido é válido usando a função *isEmail* da biblioteca *validator* e, se não for válido, adiciona outro erro à lista de erros. Outra verificação realizada é se o campo de senha está vazio e, se estiver, adiciona um erro à lista de erros indicando que o campo de senha é obrigatório. No final, a função retorna um objeto que contém a lista de erros e um valor booleano *isValid* que indica se não há erros, com base no estado da lista de erros.

O arquivo *register.js* é semelhante, mas é destinado à validação dos dados fornecidos durante o processo de registro de um novo usuário. A função

validateRegisterInput aceita um objeto chamado *data* que contém informações como nome, email, senha e confirmação de senha. Em seguida, é verificado se o campo de nome está vazio e, se estiver, é adicionado um erro à lista de erros indicando que o campo de nome é obrigatório. Além disso, se o campo de email está vazio é adicionado um erro à lista de erros indicando que o campo de email é obrigatório. Por fim é verificado se o email fornecido é válido usando a função *isEmail* da biblioteca *validator* e, se não for válido, adiciona outro erro à lista de erros. Para a senha, é necessário verificar se o campo de senha está vazio e, se estiver, é adicionado um erro à lista de erros indicando que o campo de senha é obrigatório. Além disso, se verifica se a senha possui pelo menos 6 caracteres usando a função *isLength.* No campo de confirmação de senha, se examina se o campo de confirmação de senha está vazio e, se estiver, é adicionado um erro à lista de erros indicando que o campo de confirmação de senha é obrigatório. No final, a função retorna um objeto que contém a lista de erros e um valor booleano *isValid* que indica se não há erros, com base no estado da lista de erros.

5.13 SERVIDOR *SERVER.JS*

Este arquivo desempenha um papel crucial na administração dos *scrapers* de anúncios *(ads)*, as recomendações *(recommendations)* e usuários *(users)*. Inicialmente, o código incorpora essencialmente todas as bibliotecas e módulos necessários para seu funcionamento, dentre elas destacam-se o *Express*, que serve como base para a criação da aplicação web, o *Mongoose*, utilizado para estabelecer a conexão com o banco de dados *MongoDB*, o *Body Parser*, cuja finalidade é processar as requisições HTTP, o Cors, que permite a comunicação entre domínios distintos, o *Passport*, que desempenha um papel fundamental na autenticação e, por último, o *Node-Schedule*, que viabiliza a programação de tarefas de *scraping* de anúncios.

Em seguida, o código procede à criação de uma instância do *Express* e define a porta na qual o servidor irá operar. A porta é flexível, sendo possível especificar uma porta personalizada por meio da variável de ambiente (*process.env.PORT*), ou a porta padrão 5000, se nenhuma porta for fornecida. Adicionalmente, o código configura o *Passport* para autenticação e estabelece as rotas da aplicação, as quais são rotas para gerenciamento de usuários, anúncios e

recomendações, e cada uma dessas rotas é canalizada para módulos de roteamento dedicados *(usersRouter, adsRouter e recommendationsRouter* respectivamente). Na sequência, o código estabelece uma conexão com o banco de dados MongoDB, utilizando a URL de conexão especificada em *process.env.MONGODB_URI*. Caso a conexão seja efetuada com êxito, o servidor é ativado e uma mensagem informativa é emitida no console, indicando que o servidor está em execução na porta especificada.

Após a inicialização do servidor, é invocada a função *initializeServerActions*, que desempenha ações iniciais. Essas ações consistem em verificar se já existem anúncios salvos no banco de dados. Se não for encontrado nenhum anúncio, a função inicia um processo de coleta de dados de anúncios por meio de scraping. Por outro lado, se anúncios já estiverem presentes no banco de dados, a função programa uma tarefa de scraping para ser executada periodicamente, especificamente, todos os domingos às 23:00, horário de Brasília.

Por último, a função *scheduleScrapingTask* é a responsável por agendar essa tarefa de *scraping*. No momento em que a tarefa é acionada, ela realiza a eliminação de todos os anúncios existentes no banco de dados e, na sequência, dá início ao processo de scraping por meio do controlador *adsController*.

5.14 TELAS DA APLICAÇÃO

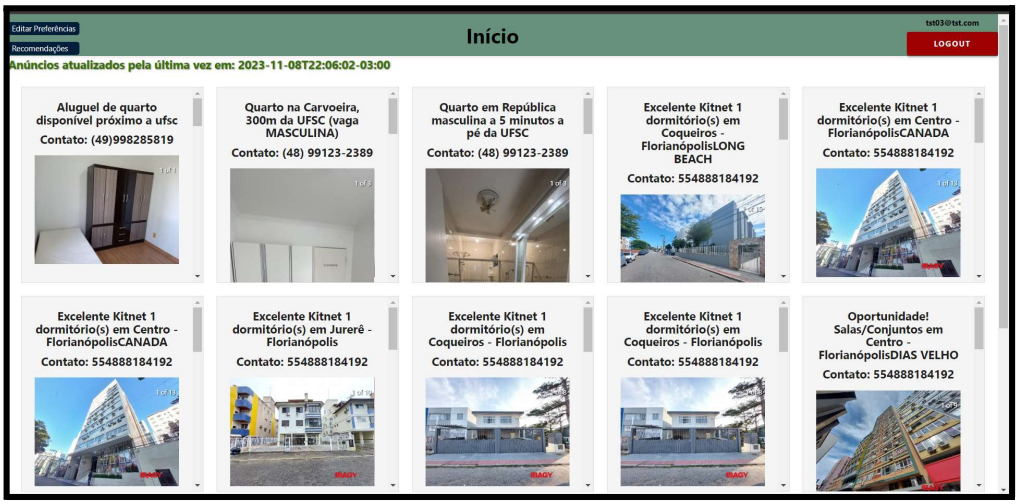
Na aplicação *‘VagouAqui’* há uma série de componentes *React* que representam as diferentes telas da aplicação. Cada componente tem seu diretório específico, ajudando a manter o código organizado e de fácil manutenção. A pasta *'AdGrid'* contém os arquivos *'AdGrid.css'* e *'AdGrid.js'*, que contém os estilos em CSS e o código JavaScript para o componente *AdGrid*, usado no aplicativo. Na pasta 'layout', encontramos arquivos de estilo e funcionalidade para a página principal do aplicativo. *'Landing.css' e 'Landing.js'* cuidam da página de entrada, *'MainPage.css' e 'MainPage.js'* tratam da página principal, enquanto *'Navbar.css' e 'Navbar.js'* são responsáveis pela barra de navegação. Finalmente, na pasta *'User',* os arquivos *'EditPreferences.css'* e *'EditPreferences.js'* provavelmente lidam com a página de edição de preferências do usuário. *'Login.css' e 'Login.js'* cuidam da página de *login*, *'Register.css' e 'Register.js'* são responsáveis pelo registro de usuários, e *'UserPage.css' e 'UserPage.js*' tratam da página do perfil do usuário.

Nas próximas seções, cada componente mencionado será abordado em detalhes e, para facilitar a compreensão, a palavra ‘componente’ será utilizada sempre que se estiver falando de uma tela da aplicação

5.14.1 COMPONENTE *ADGRID.JS*

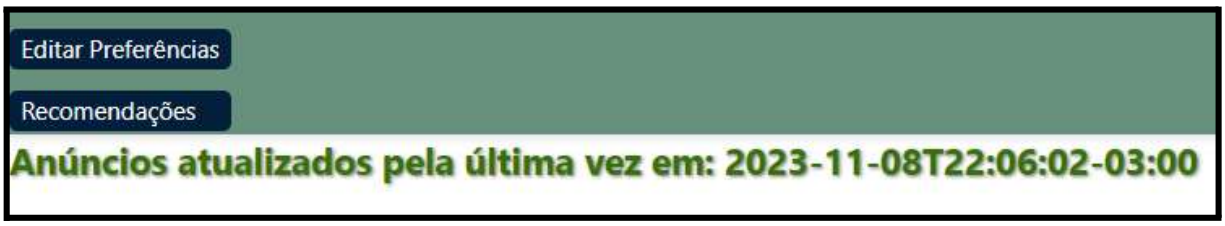
Este é o principal componente da aplicação, pois nele são exibidos todos os anúncios que estão salvos no banco de dados. O componente começa importando várias bibliotecas e módulos necessários, como *React, useState, useEffect* para gerenciar o estado e os efeitos colaterais, a biblioteca 'react-responsive-carousel' para criar um carrossel de imagens, 'react-image-lightbox' para exibir imagens em uma *lightbox*, e *'axios'* para fazer solicitações *HTTP*. Dentro da função do componente *"AdGrid"*, várias variáveis de estado são inicializadas usando o *useState*. Essas variáveis de estado incluem *"ads"* para armazenar a lista de anúncios, *"lastScrapingDate"* para a última data de raspagem, *"currentPage"* para a página atual que está sendo exibida, *"adsPerPage"* para o número de anúncios exibidos por página, *"lightboxImages"* para as imagens exibidas na lightbox e *"lightboxImageIndex"* para o índice da imagem atual na *lightbox*. A Figura 14 ilustra o componente.

Figura 14. Componente AdGrid.js com os anúncios exibidos



O *useEffect* é usado para buscar os anúncios e a última data de raspagem quando o componente é montado. Ele faz isso fazendo solicitações *HTTP* para as rotas *'/api/ads/all'* e *'/api/ads/lastScrapingDate'* usando o *Axios*. Os dados obtidos são então armazenados nas variáveis de estado *"ads" e "lastScrapingDate".* A figura 15 a seguir ilustra esse detalhe do componente.

Figura 15 Exibição da data da última atualização da base de anúncios

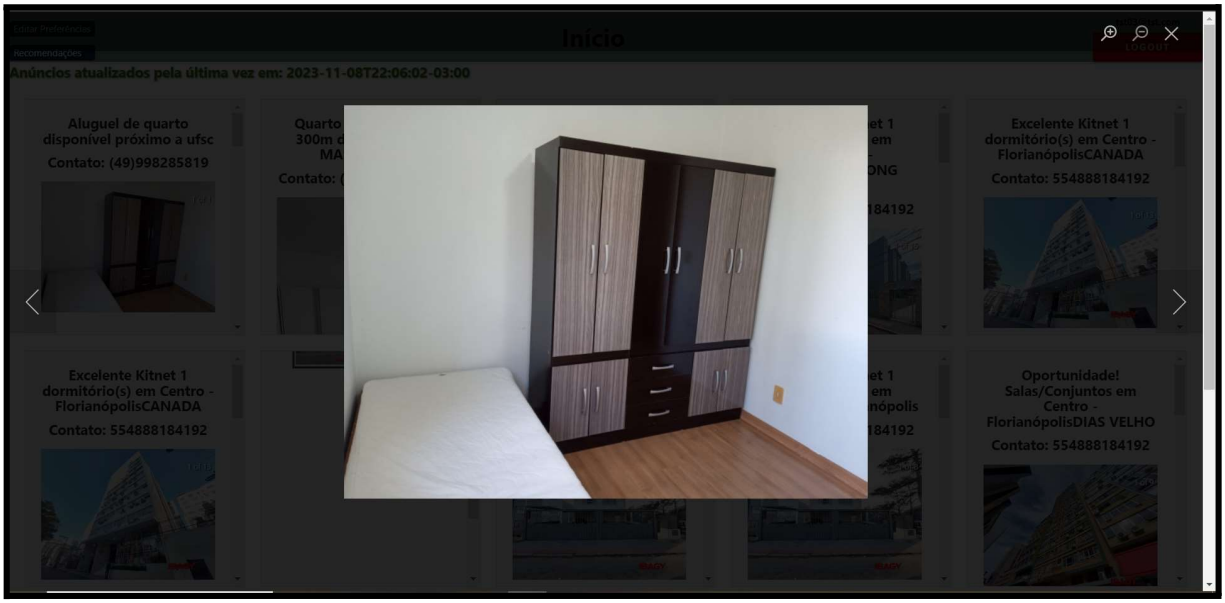


O componente determina o intervalo de páginas a serem exibidas na paginação, permitindo que o usuário navegue entre as páginas utilizando uma *"lowerBound" e "upperBound"* definidas com base na página atual e no número de páginas a serem exibidas. Esta paginação é exibida na parte inferior da grade de anúncios, permitindo ao usuário navegar entre as páginas. Existem botões *"Previous" e "Next"* para navegar para a página anterior e seguinte, bem como botões para avançar ou retroceder 10 páginas de uma só vez. A figura 16 mostra como a paginação está estruturada no *adGrid*.

Figura 16 Paginação presente no componente *AdGrid.js*

Outra funcionalidade implementada no componente ocorre quando o usuário clica em uma imagem de um anúncio: uma *lightbox* é aberta usando a biblioteca '*react-image-lightbox*', permitindo que o usuário navegue pelas imagens em um carrossel de imagens em tela cheia. A *lightbox* exibe a imagem atual, a próxima e a anterior, e o usuário pode fechá-la quando terminar de visualizar as imagens.

Figura 17 Imagem do carrossel sendo exibida utilizando o efeito *lightbox*



Cada anúncio tem a uma estrutura na seguinte ordem:

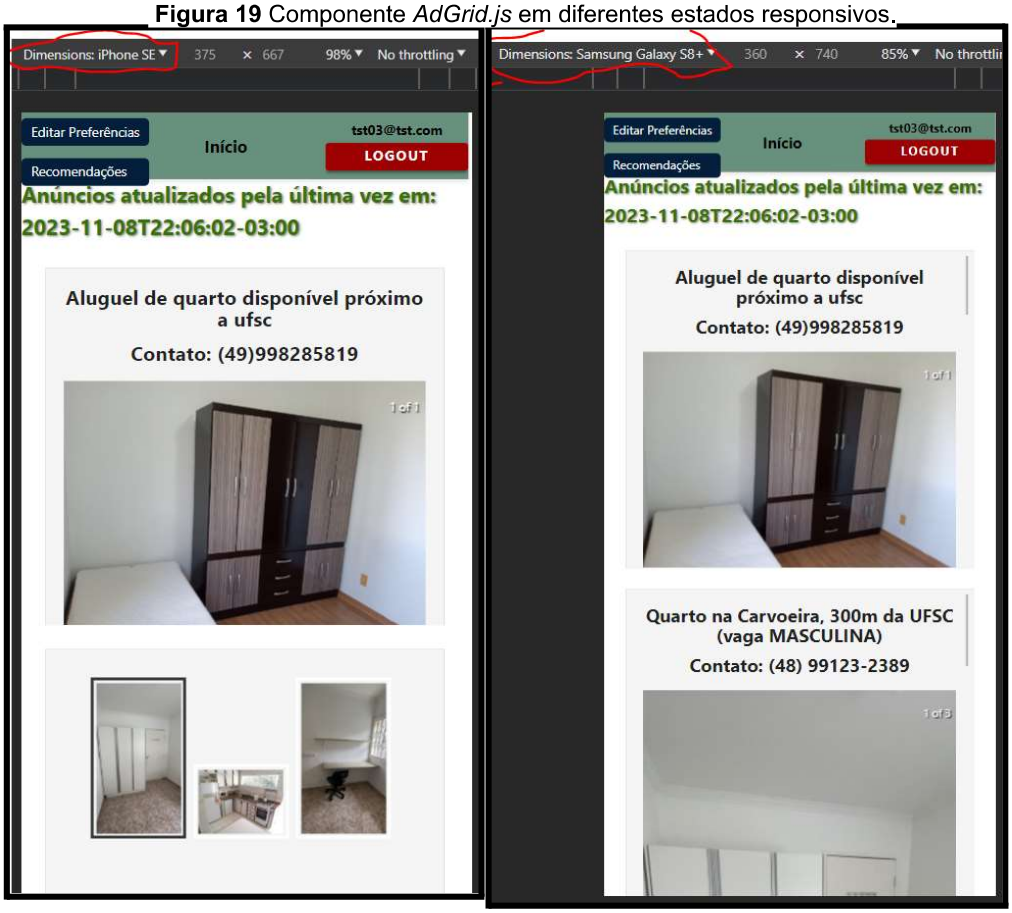
- Título do anúncio: Título original do anúncio.
- Contato: Telefone de contato. Caso nenhum telefone seja identificado, o link do anúncio original é adicionado no campo.
- Imagens do imóvel: Carrossel com todas as imagens que foram extraídas dos anúncios.
- Valor do anúncio: Valor do aluguel do anúncio.
- Bairro: Bairro no qual o imóvel está localizado.

Além desses elementos, cada anúncio também tem uma barra de rolagem para que mais registros possam ser encaixados na grade de anúncios sem prejudicar a visualização dos dados. A figura 18 mostra o como os elementos estão distribuídos:

Figura 18 Elementos de um anúncio

O componente também foi otimizado para responsividade tanto em dispositivos Android quanto dispositivos Apple, como é possível observar na Figura 19 a seguir.

Figura 19 Componente *AdGrid.js* em diferentes estados responsivos.



5.14.2 COMPONENTE *RECOMMENDEDGRID.JS*

O componente "*RecommendedGrid*" exibe uma grade de anúncios recomendados com funcionalidade de paginação, permitindo que o usuário navegue entre as páginas e visualize imagens em detalhes. Ele busca os dados de anúncios recomendados do servidor quando é montado e oferece uma experiência de visualização semelhante à do componente anterior, utilizando as mesmas tecnologias, como um carrossel de imagens e uma *lightbox* para detalhes das imagens. A Figura 20 ilustra o componente.

Figura 20 Component *RecommendedGrid.js* com os anúncios recomendados

5.14.3 COMPONENTE *LANDING.JS*

Este componente serve como página inicial para o usuário logado e exibe uma mensagem de boas-vindas personalizada para o mesmo, além de oferecer um botão para acessar a página de anúncios e interagir com o Redux para gerenciar o estado da aplicação, incluindo a funcionalidade de *logout*. Para tanto, são importadas várias bibliotecas e módulos essenciais para essas ações, incluindo o *React*, *PropTypes* (para validação de tipos de propriedades), o módulo *"connect"* (da biblioteca Redux, usado para conectar componentes ao estado global da aplicação), a ação *"logoutUser"*, o componente "Link" do *React Router* (para navegação entre páginas) e um arquivo de estilos chamado "*Landing.css*" para a estilização do componente. As Figuras 21 e 21a. ilustram o componente no modo normal e no modo responsivo, respectivamente.

Figura 21 Componente *Landing.js*

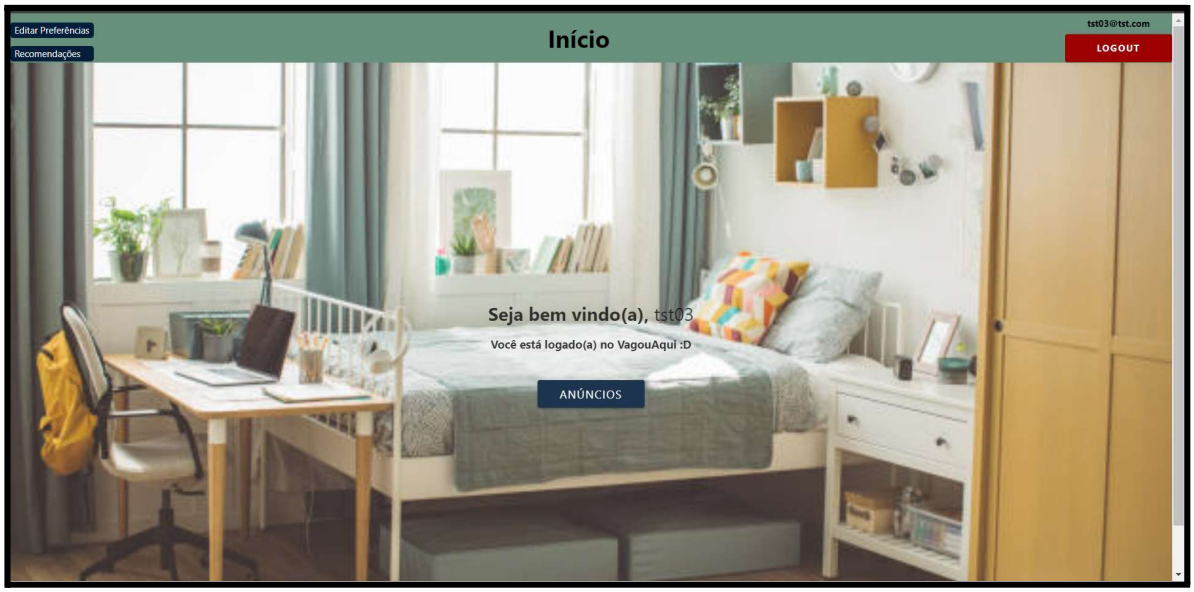


Figura 21a Componente *Landing.js* em modo responsivo

O componente *"Landing"* é declarado como uma classe que estende a classe "Component" fornecida pelo React. há uma desestruturação da propriedade "auth" para acessar informações do usuário autenticado. A desestruturação é uma técnica que torna mais fácil o acesso a partes específicas de um objeto, neste caso, o

objeto "auth" que contém informações do usuário. O trecho de código JSX contido no método de renderização é responsável por definir a estrutura da página. Inclui um container com a classe *"landing-container",* o qual possui estilos definidos no arquivo CSS importado. Dentro desse container, há uma mensagem de boas-vindas que utiliza o nome do usuário autenticado para uma saudação personalizada. Além disso, há um botão chamado "Anúncios" que é um link para a rota *"/products"* usando o componente *"Link"* do *React Router.* Isso permite que o usuário acesse a página de anúncios quando o botão é clicado. Finalmente, para garantir a consistência e a integridade do código, as *propTypes* são definidas no final do componente.

5.14.4 COMPONENTE *MAINPAGE.JS*

Este componente desempenha o papel de exibir a página inicial para qualquer pessoa que acessar a aplicação *'VagouAqui'*. No início do código, são feitas importações dos módulos essenciais, como *React, react-router-dom e react-redux*, juntamente com a inclusão de estilos CSS específicos para a página. A Figura 22 e Figura 22a. ilustram o componente no modo normal e no modo responsivo, respectivamente.

Figura 22. Component *MainPage.js* para usuários logados

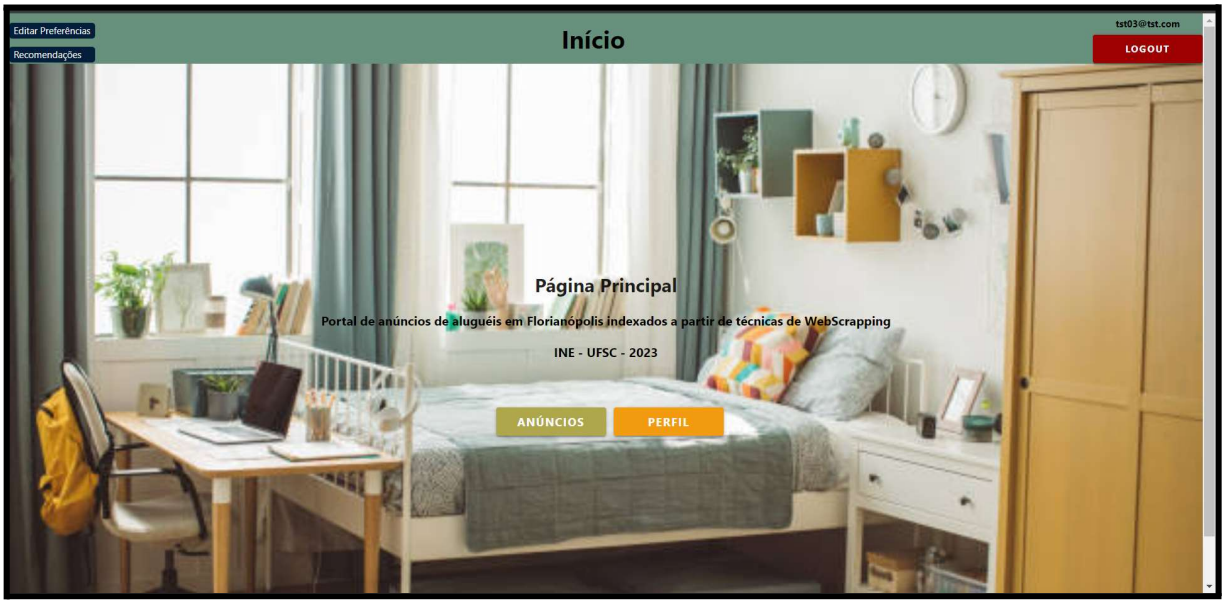


Figura 22a Component *MainPage.js* para usuários logados em modo responsivo

O *"MainPage"* é uma classe *React* que herda recursos do componente *"Component".* Essa herança possibilita que o componente seja renderizado na tela e reaja às mudanças no estado ou nas propriedades, permitindo uma apresentação dinâmica e adaptativa. O método *render* é responsável por estruturar o conteúdo da página principal, incluindo elementos HTML, como um título, uma descrição e botões de ação. Esses elementos são a base da interface do usuário, tornando-a informativa e interativa, orientando o usuário de maneira eficaz. Uma funcionalidade crítica do componente é a verificação do estado de autenticação do usuário, baseando-se nas propriedades passadas a ele. Isso permite que o componente decida o que mostrar na página com base no status de autenticação, direcionando o usuário de forma apropriada. Outro ponto interessante neste componente é o uso de condicionais ternárias para renderizar elementos diferentes, dependendo do estado de autenticação do usuário. Quando o usuário está autenticado, botões de acesso aos anúncios e ao perfil do usuário são exibidos, facilitando a navegação. Caso contrário, são mostrados botões de cadastro e login, incentivando o início de sessão ou registro, quando necessário. A Figura 23 ilustra esse comportamento.

Figura 23 Renderização dos botões baseado em ternários

Finalmente, o componente é conectado ao Redux, o que possibilita o acesso a informações relacionadas à autenticação do usuário armazenadas no estado global. Essa conexão ao estado global é essencial para garantir a sincronização entre diferentes partes da aplicação, proporcionando uma experiência coesa ao usuário.

5.14.5 COMPONENTE *NAVBAR.JS*

O componente "Navbar" foi projetado para proporcionar uma experiência de navegação prática ao usuário, pois desempenha diversas funções cruciais para a interação do usuário com a aplicação. A Figura 24 ilustra o componente.

Figura 24 Componente *Navbar.js*

Para começar, o código realiza as mesmas importações que os outros componentes mencionados, incluindo *React, react-router-dom, react-redux* e ação de *logout*. Além disso, ele incorpora um arquivo de estilo chamado *"Navbar.css"*, garantindo que a barra de navegação tenha a aparência desejada. O componente é definido como uma classe estendida da classe *"Component"* do *React*. Dentro dessa classe, se encontra a função *"onLogoutClick"*, a qual é acionada quando o usuário clica no botão *"Logout"* e tem a finalidade de evitar o comportamento padrão do clique, que geralmente recarrega a página. Em vez disso, a função *"onLogoutClick"* chama a ação *"logoutUser"* por meio do objeto *props*, permitindo ao usuário fazer logout da aplicação de forma segura.

No que diz respeito à renderização da interface, o componente verifica se o usuário está autenticado, com base nas *props* recebidas. Dependendo do status de autenticação, diferentes elementos são exibidos dinamicamente na barra de navegação, por exemplo: se o usuário estiver autenticado (logado), são apresentados links para acessar as seções "Editar Preferências" e "Recomendações". Além disso, o endereço de e-mail do usuário autenticado é exibido.

Por fim, o botão *"Logout"* é disponibilizado, que, ao ser clicado, chama a função *"onLogoutClick"* para permitir que o usuário faça logout da aplicação de forma segura. No entanto, se o usuário não estiver autenticado, a barra de navegação mostrará apenas o link para a página inicial, denominada *"Início"*. Finalmente, a capacidade de mapear o estado do Redux para o *props* do componente desempenha um papel fundamental porque permite ao componente acessar informações sobre a autenticação, como determinar se o usuário está autenticado e exibir o email do usuário, tornando a barra de navegação adaptável com base no status do usuário.

5.14.6 COMPONENTE *EDITPREFERENCES.JS*

Este componente é responsável por permitir que os usuários editem suas preferências relacionadas à busca por habitação, como aluguel, localização, tipo de moradia, preferências de gênero, aceitação de animais de estimação, acessibilidade para cadeirantes, nível de ruído, preferência por fumantes e mobília no local. A Figura 25 e Figura 25a. ilustram o componente nos modos normal e responsivo, respectivamente.

Figura 25 Componente *EditPreferences.js*

Figura 25a Componente *EditPreferences.js* em modo responsivo

Para começar, o componente é projetado para exibir um formulário de edição de preferências em um ambiente amigável ao usuário. Assim que o usuário acessa essa parte da aplicação, os campos do formulário são preenchidos com suas preferências atuais, obtidas da base de dados. O formulário permite ao usuário realizar uma série de ações importantes. Por exemplo, ele pode especificar a localização desejada para a habitação, o valor do aluguel que está disposto a pagar, se prefere casa ou apartamento, suas preferências de gênero para potenciais colegas de casa, a aceitação de animais de estimação, a acessibilidade para cadeirantes, o nível de ruído desejado, a aceitação de fumantes e se deseja uma habitação mobiliada. Ao fazer alterações em qualquer um desses campos, o usuário pode atualizar suas preferências através de um botão "Atualizar" no final do formulário. Isso envia as novas preferências para o servidor por meio de uma requisição HTTP, permitindo que as informações sejam armazenadas na base de dados.

Finalmente, o componente utiliza recursos avançados do React, como o gerenciamento de estado interno e a manipulação de eventos. Ele também se beneficia do React Router para fornecer navegação dentro da aplicação e da biblioteca Redux para gerenciar o estado global, incluindo as informações do usuário autenticado.

5.14.7 COMPONENTE *LOGIN.JS*

Este componente serve para renderizar as funcionalidades de login de um usuário. Seguindo o mesmo processo que os demais, o código importa diversas bibliotecas e módulos necessários para o funcionamento do mesmo, como o *React, React Router, PropTypes* para validação de propriedades, e o *Redux* para gerenciamento de estado. Além disso, o componente importa o arquivo *"Login.css"* para a estilização do componente. A renderização do componente também inclui um formulário com campos de entrada para email e senha, bem como áreas para exibir erros de validação. Há também links para redirecionar o usuário para a página inicial e para a página de registro, caso ainda não tenha uma conta. A Figura 26 e Figura 26a. ilustram o componente nos modos normal e responsivo, respectivamente.

Figura 26 Componente *Login.js*

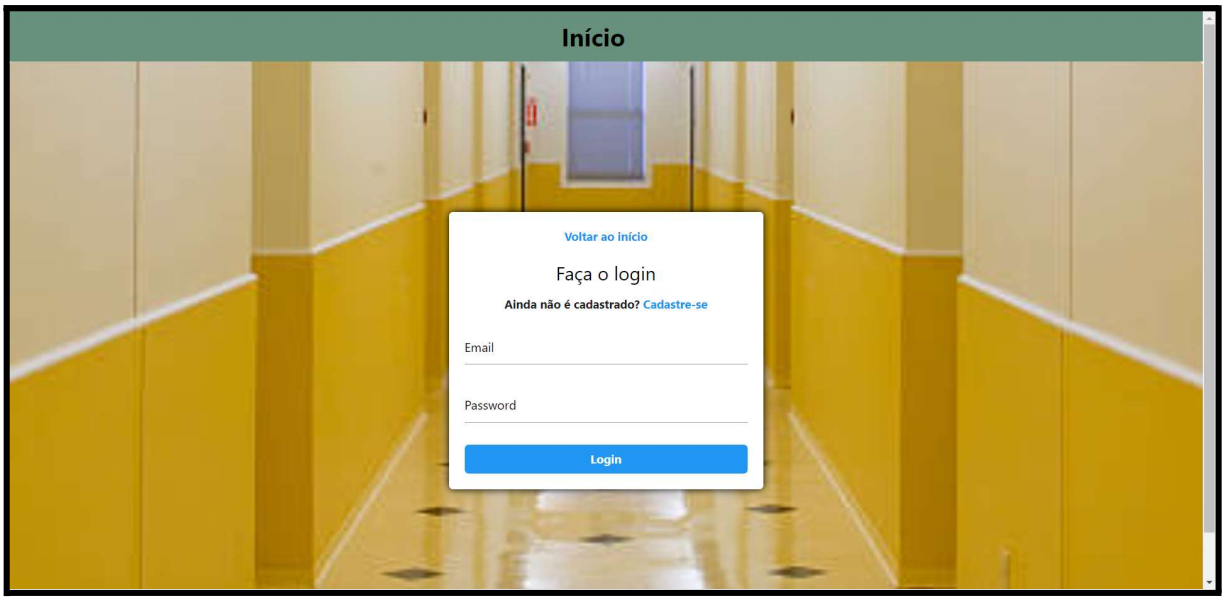
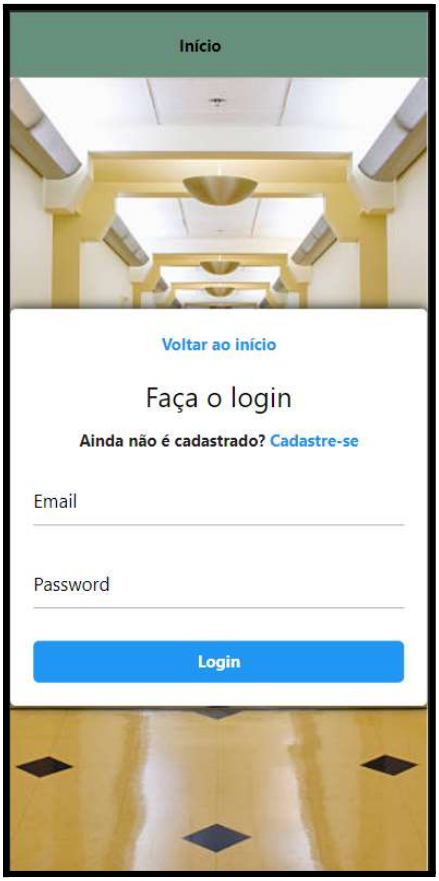


Figura 26a Componente *Login.js* em modo responsivo



Em relação ao funcionamento, o componente "Login" mantém um estado interno que armazena os dados de entrada do usuário, incluindo o email, senha e erros de validação. No método "*componentDidMount*," o componente verifica se o usuário já está autenticado. Se estiver, redireciona o usuário para a página de *dashboard*. Isso é útil para evitar que um usuário já autenticado acesse a página de login novamente. Já o método "*componentWillReceiveProps*" é usado para lidar com as alterações nas propriedades recebidas pelo componente. Se o usuário estiver autenticado, ele também redireciona para a página principal da aplicação (*Landing*). Além disso, se houver erros recebidos nas propriedades, eles são atualizados no estado do componente para serem exibidos ao usuário. Os métodos *"onChange" e "onSubmit"* tratam das interações do usuário com os campos de email e senha e do envio do formulário. Quando o usuário digita no campo de email ou senha, o estado interno é atualizado para refletir as mudanças. Quando o formulário é submetido, os dados de email e senha são coletados do estado e enviados para a função *"loginUser"* através das propriedades. Se o usuário cometer erros no

preenchimento dos campos, os erros são exibidos abaixo dos campos correspondentes. Finalmente, o componente declara suas propriedades esperadas (*propTypes*) e faz uso do Redux para conectar o estado da autenticação ("*auth*") e os erros à sua renderização. A função "*loginUser*" é fornecida como uma propriedade por meio do "*connect*" do *Redux*, permitindo que o componente despache a ação de login.

5.14.8 COMPONENTE *REGISTER.JS*

Este componente funciona como um formulário de registro de usuário dentro da aplicação *VagouAqui,* coletando informações pessoais e preferências, e as enviando para o servidor para registro. O código começa importando várias bibliotecas e componentes necessários para a aplicação, como o *React, o React Router, o Redux e o PropTypes,* bem como alguns estilos e ações de registro de usuário.

Figura 27 Componente *Register.js*

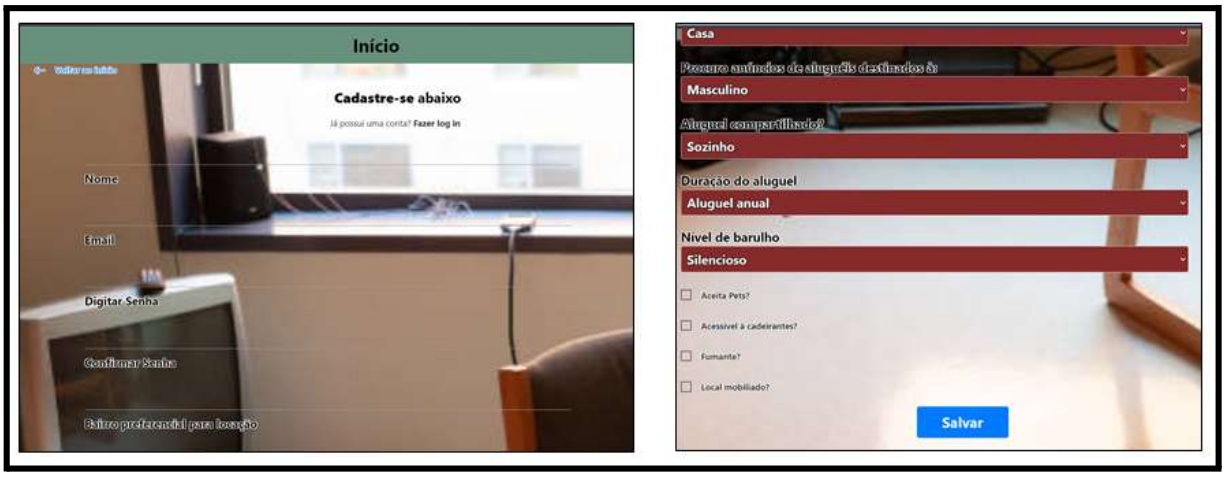
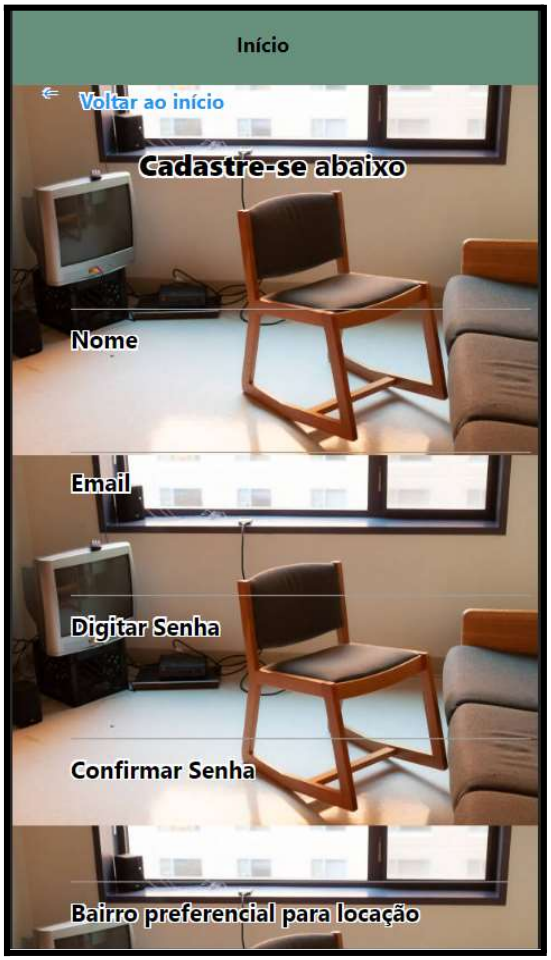


Figura 27a Componente *Register.js* em modo responsivo



Em seguida, é definida uma classe chamada "Register" que estende a classe *Component* do React. Esta classe representa a página de registro da aplicação. Dentro do construtor da classe, são definidos vários estados iniciais para o componente. Isso inclui campos relacionados ao usuário, como nome, email, senha, senha de confirmação, bem como campos relacionados às preferências do usuário, como tipo de moradia, preferência de gênero, aceitação de animais de estimação, etc.

Em relação aos métodos presentes no componente, o método "*componentDidMount*" é utilizado para redirecionar o usuário para a página de painel (dashboard) se o usuário já estiver autenticado. Já o método "*componentWillReceiveProps*" é usado para atualizar o estado do componente com possíveis erros vindos das props, que podem ser fornecidas pelo *Redux*. Também há o método *"onChange"*, o qual é responsável por atualizar o estado do componente sempre que um campo de entrada do formulário é alterado. Ele

também lida com os campos relacionados às preferências do usuário, atualizando o estado de acordo. O método "*onSubmit*" é chamado quando o formulário de registro é enviado. Ele cria um objeto "*newUser*" com as informações do usuário e suas preferências e chama a função "*registerUser*" do *Redux* para registrar o usuário. Além disso, ele usa o objeto "*history*" do *React Router* para redirecionar o usuário após o registro. O método "*render*" renderiza o formulário de registro. Ele inclui campos para nome, email, senha, senha de confirmação e uma série de campos relacionados às preferências do usuário, como localização, orçamento, tipo de moradia, etc.

O código também lida com a exibição de erros de validação e fornece botões de navegação para outras páginas, como o link "Voltar ao início" e o link "Fazer log in" componente "Register" é conectado ao Redux usando a função "connect" e as props são mapeadas para o estado do Redux, incluindo as ações necessárias, como "registerUser". Isso permite que o componente acesse os dados do estado global da aplicação.

5.14.9 COMPONENTE *USERPAGE.JS*

O componente *UserPage* é uma classe React que desempenha o papel de representar a página de perfil do usuário em um aplicativo web. Ao ser montado, o componente busca os dados do usuário, inicialmente definindo seu estado com três propriedades: *user, error e confirmDelete*. A propriedade user é utilizada para armazenar os dados do usuário logado, *error* para lidar com possíveis mensagens de erro e *confirmDelete* para controlar a exibição de um diálogo de confirmação de exclusão de conta.

Figura 28 Componente *UserPage.js*

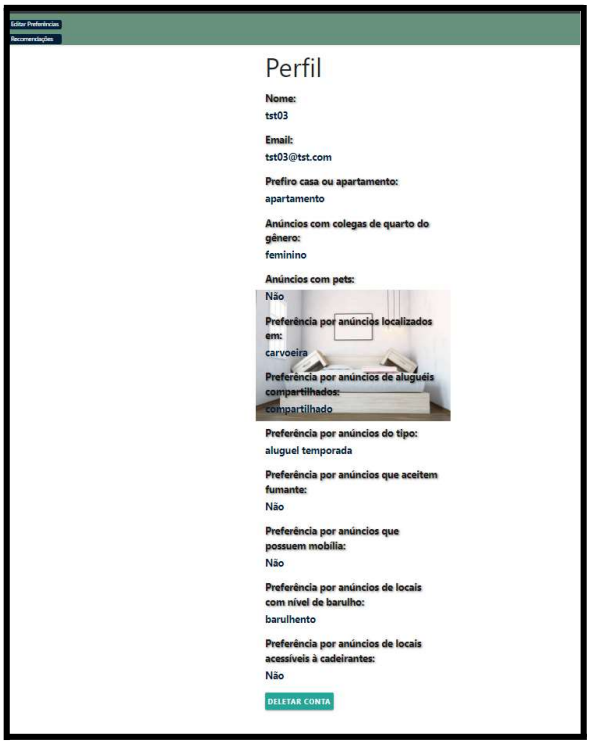
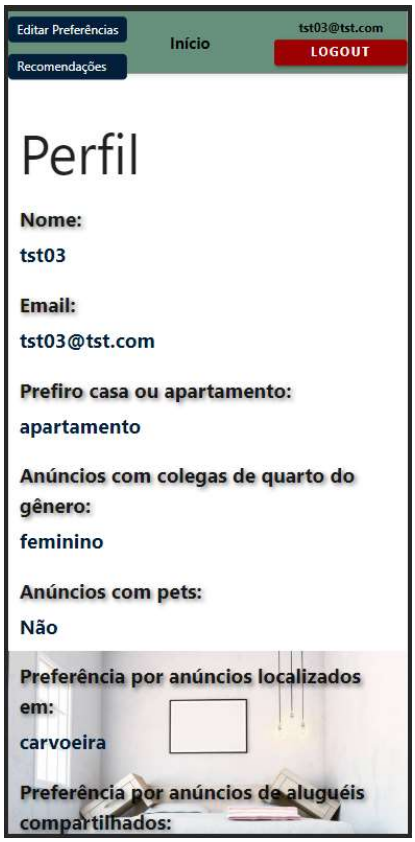


Figura 28a Componente *UserPage.js* em modo responsivo



Assim que o componente é montado, a função *componentDidMount* é acionada. Ela, por sua vez, invoca o método *fetchUserData* para buscar os dados do usuário no servidor. Através de uma solicitação GET, utilizando a biblioteca Axios, os dados do usuário são obtidos e armazenados na propriedade user, ou, em caso de falha, uma mensagem de erro é armazenada na propriedade *error*.

O componente também oferece a funcionalidade de exclusão de conta do usuário. Quando o usuário aciona a função *handleDeleteAccount*, o estado *confirmDelete* é definido como verdadeiro, o que resulta na exibição de um diálogo de confirmação para a exclusão da conta. Se o usuário confirmar a exclusão, a função *confirmDelete* é chamada. Ela efetua uma solicitação DELETE para a API, e em caso de sucesso, chama a ação *logoutUser* do *Redux* para desconectar o usuário e redirecioná-lo para a página inicial. Por outro lado, se o usuário optar por cancelar a exclusão, a função *cancelDelete* é chamada, restaurando o estado *confirmDelete* para falso. No método render, o componente exibe o perfil do usuário, incluindo informações como nome, email e preferências, caso um usuário esteja logado. Além disso, é apresentada a opção de excluir a conta, seja por meio de um diálogo de confirmação ou por um botão "Deletar conta".

Para garantir a tipagem correta das propriedades, o componente define *PropTypes* apropriadas, assegurando que *auth* seja um objeto válido. Por fim, o componente é conectado ao Redux, permitindo que ele acesse o estado do aplicativo, e a ação *logoutUser* é mapeada para as propriedades do componente. O uso de *withRouter* assegura que as propriedades de roteamento estejam disponíveis.

5.15 REDUCERS

Os *Reducers* são funções no *Redux* que especificam como o estado da aplicação é atualizado em resposta a uma ação. Eles recebem o estado atual e uma ação como argumentos e retornam um novo estado atualizado, mantendo a imutabilidade do estado. Geralmente, cada Reducer lida com uma parte específica do estado da aplicação. Eles são cruciais para gerenciar e atualizar o estado de maneira previsível e escalável. A aplicação em questão utiliza dois Reducers distintos para gerenciar efetivamente o estado. O primeiro, "*authReducer*," é responsável por lidar com a autenticação do usuário. Ele define um estado inicial

que inclui propriedades como "*isAuthenticated*," "*user*" e "*loading*." O *Reducer* responde a ações *Redux*, como "*SET_CURRENT_USER*" e "*USER_LOADING*," para atualizar o estado com base nos dados fornecidos na ação, alterando "*isAuthenticated*" e "*user*" conforme necessário. O segundo *Reducer*, "*errorReducer*," lida com erros na aplicação. Ele começa com um estado inicial vazio e responde à ação "*GET_ERRORS*" para armazenar mensagens de erro que possam ocorrer durante a execução da aplicação. O arquivo "*index*" é o ponto de entrada para combinar esses Reducers usando a função "*combineReducers*" do *Redux,* criando um único *Reducer* que incorpora os dois mencionados anteriormente. Eles são acessíveis através de propriedades distintas no estado global da aplicação, garantindo que as informações de autenticação e mensagens de erro sejam mantidas separadas e acessíveis aos componentes que delas necessitam.

5.16 ACTIONS

Os componentes da aplicação possuem algumas operações que necessitam de certos comportamentos denominados *actions*. O arquivo *'authActions.js'* realiza as ações necessárias exigidas pelos componentes no que diz respeito à autenticação de usuários e à gestão de informações do usuário. Ele contém uma série de funções com finalidades específicas, como por exemplo, a função *registerUser*, a qual é responsável por permitir o registro de novos usuários. Ela realiza uma requisição POST para um endpoint */api/users/register,* enviando os dados do usuário como parâmetro. Em caso de sucesso no registro, o usuário é redirecionado para a página de login. A função *loginUser* é utilizada para autenticar um usuário. Ela realiza uma requisição POST para o endpoint /api/users/login, enviando as credenciais do usuário. Se o login for bem-sucedido, o *token JWT* é armazenado no armazenamento local (localStorage) e as informações do usuário são decodificadas a partir desse token, o qual faz com que o usuário seja considerado autenticado. Para atualizar o estado da aplicação com as informações do usuário autenticado, utiliza-se a função *setCurrentUser*, a qual recebe um objeto decodificado a partir do *token JWT* e o utiliza para atualizar o estado do usuário logado.

A função *setUserLoading* tem a finalidade de indicar que a aplicação está em processo de carregamento de informações do usuário, sendo útil para exibir indicadores de carregamento. Por outro lado, a função *logoutUser* é responsável por efetuar o *logout* do usuário. Ela remove o *token JWT* do armazenamento local, elimina o cabeçalho de autenticação das requisições futuras (usando *setAuthToken*) e redefine o estado do usuário como vazio, indicando que o usuário não está mais autenticado. Finalmente, a função *updateUserPreferences* é utilizada para a atualização das preferências do usuário. Ela envia uma requisição PUT para o endpoint */api/users/preferences*, transmitindo as preferências atualizadas. Se a atualização for bem-sucedida, o usuário é redirecionado para a página de produtos.

5.17 COMPONENTES AUXILIARES

Da mesma forma que ocorreu no Servidor, o Cliente também utiliza uma série de arquivos auxiliares para tornar os componentes da aplicação mais seguros. Esses arquivos desempenham funções específicas e, juntos, permitem que a aplicação forneça uma experiência segura e protegida para os usuários, garantindo que apenas aqueles que estão autenticados tenham acesso a certas partes da aplicação.

O arquivo *setAuthToken.js* é responsável por gerenciar a autenticação. Ele utiliza a biblioteca Axios para incluir o token de autenticação nas requisições HTTP. Quando um token é fornecido, o cabeçalho de autorização é configurado; caso contrário, é removido. O arquivo *PrivateRoute.js* cria rotas protegidas acessíveis apenas por usuários autenticados. Ele utiliza o React Router para verificar a autenticação do usuário e redirecioná-lo para a página de login se necessário. No caso do arquivo *index.js*, ele é o ponto de entrada da aplicação *VagouAqui*. A função *ReactDOM.render* presente nele é usada para renderizar o conteúdo da aplicação no *HTML*, colocando o componente *<App />* no elemento com o *ID "root"* do documento *HTML*. Finalmente, o arquivo *index.html* é a página principal da aplicação, definindo a estrutura básica dela. A finalidade deste arquivo é criar a interface do usuário da aplicação fornecendo um elemento , onde a aplicação ‘*VagouAqui*’ será renderizada.

5.18 REDUX STORE

O arquivo *store.js* tem a finalidade de criar uma *Redux store* (armazém *Redux* em tradução livre), a qual é essencial para aplicações que buscam um gerenciamento de estado consistente e escalável. A sua função principal é centralizar o armazenamento e a gestão do estado global da aplicação.

Primeiramente, são importadas algumas funções essenciais do universo *Redux*, como, *createStore, applyMiddleware e compose,* as quais provêm as bases para a configuração da loja. Além disso, é importado o *middleware thunk* da biblioteca *"redux-thunk",* o qual é uma ferramenta fundamental que permite a criação de ações assíncronas no *Redux*. Por fim, o *rootReducer* é importado, o qual contém os *reducers* combinados da aplicação.

O ponto de partida do estado da aplicação é representado pelo *initialState*, o qual é definido inicialmente como um objeto vazio.. A partir daqui, é possível adicionar os valores iniciais que se deseja que a aplicação utilize. O próximo passo é a configuração do *middleware*, a qual utiliza *o middleware thunk*, e o aplica à loja Redux. A criação dela ocorre com a função *createStore*, a qual utiliza três argumentos: o *rootReducer*, que determina como o estado da aplicação é atualizado em resposta a ações; o *initialState*, que é o estado inicial de nossa aplicação; e o *compose*, que é uma função usada para combinar diversos aprimoradores da loja, como *middleware* ou a extensão *Redux DevTools*.

Um aspecto importante é a integração da extensão *Redux DevTools*. Se a *window.__REDUX_DEVTOOLS_EXTENSION__* estiver disponível, geralmente devido a uma extensão instalada no navegador, ela é utilizada para melhorar a loja com recursos do *DevTools*. Caso contrário, é empregada uma função simples f => f como alternativa. Isso assegura que o código continue funcionando mesmo se a extensão DevTools não estiver presente ou ativa.

Por fim, a loja é criada com todas as configurações mencionadas e é exportada como a exportação padrão deste módulo. Isso possibilita que outras partes da aplicação importem a loja e a utilizem para gerenciar o estado da aplicação.

5.19 ARQUIVO APP.JS

Este arquivo serve como o ponto central de configuração e estruturação da interface da aplicação, especificamente em uma aplicação desenvolvida em React, garantindo o gerenciamento de rotas, estado global, autenticação de usuário e a renderização de componentes globais.

Dentro do código, existe um bloco condicional que verifica a presença de um *token JWT* armazenado no *localStorage*. Se ele for detectado, o código executa uma série de ações importantes: Primeiramente, configura o *token JWT* como um cabeçalho de autenticação para ser utilizado em futuras solicitações *HTTP*, o que é realizado através da função "*setAuthToken*". A decodificação é feita para obter informações sobre o usuário e o tempo de expiração. Posteriormente, o código define o usuário atual e o marca como autenticado usando a função *"setCurrentUser"*. Além disso, o código verifica se o *token JWT* expirou, comparando a data de expiração com o horário atual. Caso o token esteja expirado, o usuário é deslogado e redirecionado para a página de login. O token tem validade de 5 minutos.

Dentro da classe "App", é onde ocorre a estruturação da aplicação pois ela envolve todo o conteúdo com um componente "*Provider*," responsável por gerenciar o estado global da aplicação por meio do Redux. Na sequência, é definido um roteador (*Router*) que controla as diversas rotas da aplicação, pois dentro do componente "App," se encontram várias rotas definidas usando o componente *"Route"* do *React Router*. Essas rotas correspondem a diferentes partes da aplicação, como a página inicial, o registro, o login e outras páginas relevantes. Adicionalmente, há um componente *"Switch"* que assegura que apenas uma rota seja correspondida por vez. É importante mencionar que algumas rotas são marcadas como *"PrivateRoute"* indicando que somente podem ser acessadas se o usuário estiver autenticado. Se o usuário não estiver autenticado e tentar acessar uma rota privada, o código realiza o redirecionamento para a página de login.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este projeto culminou na criação da aplicação web *"VagouAqui"*, destinada a facilitar a vida daqueles que precisam de um local para morar. Apesar de ter um foco maior em estudantes da UFSC, por isso a maior quantidade de anúncios de quartos para alugar em locais compartilhados em locais próximos à UFSC, o processo de agregação de anúncios reúne cerca de 1000 anúncios no mínimo toda vez que é executado. Sendo assim há variedade o suficiente para que qualquer pessoa que esteja de mudança consiga fazer bom proveito da aplicação. Além disso, esse trabalho serviu como um grande exercício dos mais variados conceitos de programação e gerência de projetos que foram aprendidos durante a graduação. O desenvolvimento tentou aderir sempre que possível aos princípios do padrão MVC de programação, além de utilizar técnicas de otimização de código sempre que possível.

Sendo a primeira experiência na qual eu construí uma aplicação do começo ao fim, durante vários momentos do projeto foram enfrentados diferentes desafios, como curvas de aprendizado tecnológico, as quais foram superadas com a ajuda de pesquisa extensiva, além do gerenciamento eficaz do projeto. Os casos de uso criados na Seção 5.1 foram utilizados como guia para que não se perdesse de vista quais eram os requisitos do sistema e não criar várias funcionalidades extras e esquecer do principal.

Durante a criação dos componentes *React* foi identificada a necessidade de deixar todas as telas responsivas, visto que isso faria com que mais pessoas utilizassem a aplicação, já que o dispositivo móvel é o principal meio de acesso à internet do brasileiro. Essa etapa definitivamente foi a mais desafiadora, pois as tecnologias empregadas eram novas para mim, então o aprendizado ocorreu com bastante tentativa e erro. A etapa de desenvolvimento do servidor e dos *scrapers* ocorreu, em grande parte do tempo, sem problemas. O único contratempo identificado foi que, à medida que mais *scrapers* eram adicionados à aplicação, o processo de coleta de dados ficava lento. Foram tentadas várias mudanças no código, até que houve a introdução de *Promises* no processo de *scraping*. Essa funcionalidade fez com que o tempo de raspagem de anúncios caísse de 40 minutos para 5 minutos. A biblioteca *node-cron* também deve ser mencionada como um

ponto de relevância no projeto, por ser uma solução para agendamento de tarefas que requer poucos passos para configuração e uso. O sistema de recomendação representa uma implementação simplificada de um sistema baseado em filtragem de conteúdo, o que é recomendado para o caso de uso da aplicação.

Deixo registrado como proposta para trabalhos futuros, a implementação do módulo administrador de anúncios para que os usuários possam cadastrar, editar e excluir anúncios individuais. Além disso, originalmente, a etapa final do trabalho era migrar a aplicação para a nuvem AWS, mas, devido à falta de tempo e prática na configuração de serviços em uma nuvem, especificamente a criação e configuração de sub-redes e grupos de segurança, a implantação não ocorreu e ficou registrada como sugestão. Num segundo momento, recomendo a expansão do componente dos anúncios para incorporar campos de filtragem, além de oferecer uma melhor experiência para o usuário, Finalmente, também proponho o uso de uma biblioteca adicional de *scraping* que consiga lidar com *modals*, o que pode expandir a quantidade de informações que cada anúncio pode ter e proporcionar ainda mais opções para os usuários.

REFERÊNCIAS

FORBES. **As 10 melhores universidades do Brasil em 2022**. Forbes, [S.l.], 2022. Disponível em: https://forbes.com.br/carreira/2022/08/as-10-melhores-universidades-do-brasil-em-2022/. Acesso em: 14 mar. 2023.

REVISTA ISTO É. **As Melhores cidades do Brasil em 2022**. Revista Isto É, [S.l.], 27 jun. 2022. Edição Especial. Disponível em: <https://www.austin.com.br/Midia/27-06-2022%20As%20Melhores%20Cidades%20do%20Brasil%20-%202022%20-%20Edi%C3%A7%C3%A3o%20Especial%20(Revista%20ISTO%C3%89)/10363>. Acesso em: 14 mar. 2023.

V PESQUISA NACIONAL DE PERFIL SOCIOECONÔMICO E CULTURAL DOS GRADUANDOS DAS IFES - 2018. Associação Nacional dos Dirigentes das Instituições Federais de Ensino Superior (ANDIFES), [S.l.], 2018. Disponível em: <https://www.andifes.org.br/?p=79639>. Acesso em: 14 mar. 2023.

CORDEIRO, Alexander Magno et al. **Revisão sistemática: uma revisão narrativa**. Revista do Colégio Brasileiro de Cirurgiões, v. 34, p. 428-431, 2007. Disponível em: <https://www.researchgate.net/publication/250989402_Revisao_sistematica_uma_revisao_narrativa/fulltext/569859f708aec79ee32b81d5/Revisao-sistematica-uma-revisao-narrativa.pdf>. Acesso em: 12 mar. 2023.

ZHAO, Bo. **Web scraping. In: Encyclopedia of big data**. [S.l.]: Springer, 2017. p. 1-3. RESNICK, Paul; VARIAN, Hal R. Recommender systems. Communications of the ACM, v. 40, n. 3, p. 56-58, 1997.

GELLERSEN, H.-W.; GAEDKE, Martin. **Object-oriented web application development**. IEEE Internet Computing, v. 3, n. 1, p. 60-68, 1999.

TAURION, Cezar. **Cloud computing-computação em nuvem**. Brasport, 2009.

SOUSA, Flávio RC; MOREIRA, Leonardo O.; MACHADO, Javam C. **Computação**

em nuvem: Conceitos, tecnologias, aplicações e desafios. II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI), p. 150-175, 2009.

SANTOS, Augusto Carvalho dos; PEDROSA, Bruno Gil; SANTOS, Henrique; GODINHO, Higor Gonçalves; SILVA, Leandro; SCHETTINO, Rafael do Carmo; SANTOS, Rodrigo Carvalho dos; MENDES, Sérgio Henrique Amaral. **Computação em nuvem: Conceitos e Perspectivas.** 2009. Belo Horizonte: IETEC – Instituto de Educação Tecnológica.

BHARDWAJ, Sushil; JAIN, Leena; JAIN, Sandeep. **Cloud computing: A study of infrastructure as a service (IAAS)**. International Journal of engineering and information Technology, v. 2, n. 1, p. 60-63, 2010.

MATHEW, Sajee; VARIA, J. **Overview of amazon web services**. Amazon Whitepapers, v. 105, p. 1-22, 2014.

AMAZON WEB SERVICES. **Amazon EC2**. Disponível em: <https://aws.amazon.com/ec2/>. Acesso em: 18 abr. 2023.

PATTERSON, Scott. **Learn AWS Serverless Computing: A Beginner's Guide to Using AWS Lambda, Amazon API Gateway, and Services from Amazon Web Services**. Packt Publishing Ltd, 2019.

AMAZON WEB SERVICES. **Amazon Lambda AWS**. Disponível em: <https://aws.amazon.com/pt/lambda/>. Acesso em: 18 abr. 2023.

AMAZON WEB SERVICES. **Provisionamento de Infraestrutura como código - AWS**. Disponível em: <https://aws.amazon.com/pt/cloudformation/>. Acesso em: 18 abr. 2023.

SAHU, Saurabh et al. Sentimental Analysis on Web Scraping Using Machine Learning Method. **Journal of Information and Computational Science (JOICS)**, ISSN, p. 1548-7741.

The industry standard for working with HTML in JAvaScript. **Cheerio.** Disponível em

<https://cheerio.js.org/>. Acesso em: 20 de outubro de 2023.

DA MOTTA, Claudia Lage Rebello et al. **Sistemas de recomendação.** 2012.

CHAFFER, Jonathan. **Learning jQuery**. Packt Publishing Ltd, 2013.

Wong, C.-I & Wong, Kin Yeung & Ng, K.-W & Fan, W. & Yeung, Alan. (2014). **Design of a crawler for online social networks analysis**. WSEAS Transactions on Communications. 13. 263-274.

LUCIANO, Josué; ALVES, Wallison Joel Barberá. **Padrão de arquitetura MVC: Model-view-controller**. EPeQ Fafibe, v. 1, n. 3a, p. 102-107, 2017.

CHAUHAN, Anjali. **A review on various aspects of MongoDB databases**. International Journal of Engineering Research & Technology (IJERT), v. 8, n. 05, p. 90-92, 2019.

MARDAN, Azat. **Express. js Guide: The Comprehensive Book on Express. js**. Azat Mardan, 2014.

CANTELON, Mike et al. **Node. js in Action**. Greenwich: Manning, 2014.

RFC 7519 - JSON Web Token. RFC. Disponível em: <https://datatracker.ietf.org/doc/html/rfc7519>. Acesso em: 20 de outubro de 2023.

APÊNDICE - CÓDIGO FONTE

CAMINHO: server/server.js:

ARQUIVO: server.js

```
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const cors = require("cors");
const passport = require("passport");
const schedule = require("node-schedule");
const adsRouter = require("./routes/api/ads");
const recommendationsRouter = require("./routes/api/recommendation");
const adsController = require("./controller/adsController");
const usersRouter = require("./routes/api/users");
const Ad = require("./models/Ads");

const app = express();
const port = process.env.PORT || 5000;

require("dotenv").config();
require("./config/passport")(passport);

app.use(cors());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(passport.initialize()); // Middleware do Passport
app.use("/api/users", usersRouter); // Rota de Usuários
app.use("/api/ads", adsRouter); // Rota de Anúncios
app.use("/api/recommendation", recommendationsRouter); // Rota de
Recomendações

mongoose
  .connect(process.env.MONGODB_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => {
    console.log("Conexão bem sucedida com o MongoDB");
    // Inicia o servidor Express
    app.listen(port, () => {
      console.log(`O servidor está rodando na porta: ${port}!`);
      // Após o servidor ser iniciado, realiza outras ações
      initializeServerActions();
    });
  })
  .catch((error) => {
    console.error("Erro de conexão com o banco de dados:", error);
  });
```

```
// Agenda a tarefa para ser executada todo domingo às 23:00 (horário de Brasília)
function scheduleScrapingTask() {
    schedule.scheduleJob("0 23 * * 0", async function () {
        try {
            // Remove anúncios existentes da coleção 'ads'
            await Ad.deleteMany({});
            console.log("Base atual de anúncios foi excluída com sucesso.");

            // Inicia o processo de scraping
            await adsController.startScraping();
        } catch (error) {
            console.error("Erro durante o processo de scraping:", error);
        }
    });
}

// Função para realizar outras ações após o servidor ser iniciado
async function initializeServerActions() {
    try {
        // Verifica se existem anúncios salvos no banco de dados
        const existingAdsCount = await Ad.countDocuments();

        if (+existingAdsCount === 0) {
            // Se não houver anúncios salvos, inicia o processo inicial de scraping
            console.log("Nenhum anúncio encontrado no banco de dados. Iniciando o
processo inicial de scraping.");
            await adsController.startScraping();
        } else {
            // Se existirem anúncios no banco de dados, segue a tarefa agendada de
scraping
            console.log("Foram encontrados anúncios no banco de dados. Seguindo a
programação do processo de scraping.");
            scheduleScrapingTask();
        }
    } catch (error) {
        console.error("Erro durante a verificação de anúncios existentes:", error);
    }
}
```

CAMINHO: server/validation/register.js
ARQUIVO: register.js

```
const Validator = require("validator");
const isEmpty = require("is-empty");

module.exports = function validateRegisterInput(data) {
    let errors = {};

    // Converte campos vazios para uma string vazia, para que possamos usar as
funções do validador
    data.name = !isEmpty(data.name) ? data.name : "";
    data.email = !isEmpty(data.email) ? data.email : "";
    data.password = !isEmpty(data.password) ? data.password : "";
    data.password2 = !isEmpty(data.password2) ? data.password2 : "";

    // Verificações do nome
    if (Validator.isEmpty(data.name)) {
        errors.name = "O campo Nome é obrigatório";
    }

    // Verificações do email
    if (Validator.isEmpty(data.email)) {
        errors.email = "O campo Email é obrigatório";
    } else if (!Validator.isEmail(data.email)) {
        errors.email = "O campo Email está com informação inválida";
    }

    // Verificações da senha
    if (Validator.isEmpty(data.password)) {
        errors.password = "O campo Senha é obrigatório";
    }

    if (Validator.isEmpty(data.password2)) {
        errors.password2 = "O campo Confirmar senha é obrigatório";
    }

    if (!Validator.isLength(data.password, { min: 6, max: 30 })) {
        errors.password = "A Senha deve conter pelo menos 6 caracteres";
    }

    if (!Validator.equals(data.password, data.password2)) {
        errors.password2 = "As Senhas devem ser iguais";
    }

    return {
        errors,
        isValid: isEmpty(errors)
    };
};
```

CAMINHO: server/scrapers/webQuartoScraper.js
ARQUIVO: webQuartoScraper.js

```
const axios = require('axios');
const cheerio = require('cheerio');
const { extractContactInfoFromDescription } = require('./contactInfoScraper');
const axiosRateLimit = require('axios-rate-limit');

// Cria uma instância do axios com limitação de taxa
const axiosInstance = axiosRateLimit(axios.create(), {
    maxRequests: 1, // Número de requisições por segundo
    perMilliseconds: 1000, // Milissegundos por requisição (neste caso, 1 requisição
por segundo)
});

// Define uma função para fazer scraping de uma única página do WebQuarto
async function scrapeWebQuartoadsPage(pageNumber) {
    try {
        const url =
`https://www.webquarto.com.br/busca/quartos/florianopolis-sc?page=${pageNumber}
`;
        const response = await axiosInstance.get(url); // Usa a instância do axios com
limitação de taxa

        if (response.status === 200) {
            const $ = cheerio.load(response.data);
            const ads = [];
            const adsPage = $('body > script:nth-child(12)');
            const scriptHtml = adsPage.html();
            const jsonString = scriptHtml.substring(scriptHtml.indexOf('{'));
            const targetString = 'window.search.city_name = \'Florianópolis - SC\';';
            const startIndex = jsonString.indexOf(targetString);
            const truncatedHtml = jsonString.substring(0, startIndex);
            const secondIndex = truncatedHtml.indexOf(';\n' + '        ');
            const finalJson = JSON.parse(truncatedHtml.substring(0, secondIndex));

            for (let i = 0; i < finalJson.ads.length; i++) {
                const ad = finalJson.ads[i];
                const contactInfo = extractContactInfoFromDescription(ad.description);
                const imageLinks = ad.photos.map(photo => photo.url);

                ads.push({
                    title: ad.title,
                    link: ad.url,
                    description: ad.description,
                    price: ad.rent_price,
                    imageLinks: imageLinks,
                    neighborhood: ad.district,
                    contactInfo: contactInfo.length === 0 ? ad.url : contactInfo
```

```
            });
        }

        return ads;
    } else {
        console.error(`Erro ao listar o json de dados para a página ${pageNumber}.
Código do status: ${response.status}`);
    }
  } catch (error) {
    console.error(`Erro durante o scraping da página ${pageNumber}:`,
error.message);
  }
}

// Define uma função para fazer scraping de todas as páginas de 1 a 8
async function scrapeWebQuartoads() {
  const allAds = [];
  for (let pageNumber = 1; pageNumber <= 8; pageNumber++) {
    const ads = await scrapeWebQuartoadsPage(pageNumber);
    if (ads) {
      allAds.push(...ads);
    }
  }
  return allAds;
}

module.exports = {
  scrapeWebQuartoads
};
```

CAMINHO: server/scrapers/vivaRealScraper.js
ARQUIVO: vivaRealScraper.js

```
const axios = require('axios');
const cheerio = require('cheerio');
const axiosRateLimit = require('axios-rate-limit');
const { extractContactInfoFromDescription } = require("./contactInfoScraper");

// Cria uma instância do axios com limitação de taxa (1 requisição por segundo)
const axiosInstance = axiosRateLimit(axios.create(), {
  maxRequests: 1, // Número de requisições por segundo
  perMilliseconds: 1000, // Milissegundos por requisição (neste caso, 1 requisição
por segundo)
});

// Função para fazer scraping dos links de imagem da página de anúncio do
VivaReal
async function extractVivaRealImageLinks(adLink) {
  try {
```

```
        const response = await axiosInstance.get(adLink);

        if (response.status === 200) {
            const $ = cheerio.load(response.data);

            // Encontra o conteúdo principal da página de anúncio
            const imageElements = $('.carousel__image');

            // Extrai os links de imagem do atributo src
            const imageLinks = imageElements.map((index, element) =>
$(element).attr('src')).get();

            return imageLinks;
        } else {
            console.error(`Erro ao realizar o scraping dos links de imagem do anúncio:
${adLink}. Código do status: ${response.status}`);
            return [];
        }
    } catch (error) {
        console.error(`Erro ao realizar o scraping dos links de imagem do anúncio:
${adLink}`, error.message);
        return [];
    }
}

// Função para fazer scraping dos detalhes de anúncios no VivaReal
async function getVivaRealAdLinks() {
    try {
        const response = await
axios.get('https://www.vivareal.com.br/aluguel/santa-catarina/florianopolis/kitnet_resi
dencial/');

        if (response.status === 200) {
            const $ = cheerio.load(response.data);

            // Encontra o conteúdo principal do anúncio
            const adList = $('.js-card-selector');

            // Cria um array para armazenar promessas para detalhes e links de imagem
do anúncio
            const adDetailsPromises = adList.map(async (index, element) => {
                const adTitleElement =
$(element).find('a.property-card__content-link.js-card-title');
                const adTitle = adTitleElement.text().trim();
                const adLink = 'https://www.vivareal.com.br' + adTitleElement.attr('href');
                const adPrice = $(element).find('.property-card__price').text().trim();
                const address = $(element).find('.property-card__address').text().trim();

                // Usa await para resolver a promessa da descrição
                const adDescription = await extractDescription(adLink);
```

```
            const contactInfo = extractContactInfoFromDescription(adDescription);
            // Extrai o bairro usando a função extractNeighborhood
            const neighborhood = extractNeighborhood(address);

            // Busca links de imagem e detalhes do anúncio em paralelo
            const [imageLinks, adDetails] = await Promise.all([
                extractVivaRealImageLinks(adLink),
                Promise.resolve({
                    title: adTitle,
                    description: adDescription,
                    link: adLink,
                    price: adPrice,
                    neighborhood: neighborhood,
                    contactInfo: contactInfo.length === 0 ? adLink : contactInfo
                }),
            ]);

            return { ...adDetails, imageLinks };
        }).get();

        // Aguarda a resolução de todas as promessas de detalhes do anúncio
        const adDetailsArray = await Promise.all(adDetailsPromises);
        return adDetailsArray;
    } else {
        console.error(`Erro ao buscar detalhes de anúncios. Código do status:
${response.status}`);
    }
  } catch (error) {
    console.error('Erro durante o scraping de informações sobre um anúncio:',
error.message);
  }
}

async function extractDescription(adLink) {
  try {
    const response = await axios.get(adLink);

    if (response.status === 200) {
        const $ = cheerio.load(response.data);
        const adDescription = $('#js-site-main > div.main-container >
div.main-features-container > div.details-container > div.description > div > div >
p').text().trim();
        return adDescription;
    }
  } catch (error) {
    console.error('Erro ao realizar o scraping da descrição do anúncio:',
error.message);
  }
}
```

```
// Função para extrair o bairro do endereço
function extractNeighborhood(address) {
    // Divide o endereço por vírgulas e hifens
    const parts = address.split(/,|-/);

    // Encontra a primeira parte não vazia que representa o bairro
    for (let i = 0; i < parts.length; i++) {
        const trimmedPart = parts[i].trim();

        // Verifica se a parte contém apenas letras ou espaços (potencial bairro)
        if (/^[A-Za-z\s]+$/.test(trimmedPart)) {
            return trimmedPart;
        }
    }

    // Se nenhum bairro for encontrado, retorna null
    return null;
}

module.exports = {
    getVivaRealAdLinks,
};
```

CAMINHO: server/scrapers/mgfScraper.js
ARQUIVO: mgfScraper.js

```
const axios = require('axios');
const cheerio = require('cheerio');
const { extractContactInfoFromDescription } = require("./contactInfoScraper");

// Função para fazer scraping e exibir valores de href da página de anúncios do
MGF
async function extractMgfHrefValues() {
    try {
        const baseUrl =
'https://www.mgfimoveis.com.br/aluguel/kitnet/sc-florianopolis?page=';
        const adLinks = new Set(); // Usa um Set para armazenar URLs únicos

        const getPageLinks = async (pageNumber) => {
            const url = `${baseUrl}${pageNumber}`;
            const response = await axios.get(url);

            if (response.status === 200) {
                const $ = cheerio.load(response.data);
                const adList = $('#slist > div');

                adList.children().each((index, element) => {
                    const adLink = $(element).find('a.h-100.d-flex.flex-column').attr('href');
                    if (adLink) {
```

```
                    adLinks.add(adLink); // Adiciona a URL ao Set para garantir
unicidade
                }
            });
        }
    };

    const pagePromises = [];
    for (let pageNumber = 1; pageNumber <= 30; pageNumber++) {
        pagePromises.push(getPageLinks(pageNumber));
    }

    await Promise.all(pagePromises);

    const adLinksArray = Array.from(adLinks); // Converte Set para um array
    const adDetails = await extractMgfAdDetails(adLinksArray);

    return adDetails;
  } catch (error) {
    console.error('Erro durante o scraping do valor de href para os anúncios:',
error.message);
  }
}

async function extractMgfAdDetails(adLinks) {
  const adDetailPromises = adLinks.map(async (adLink) => {
    try {
      const response = await axios.get(adLink);
      if (response.status === 200) {
        const $ = cheerio.load(response.data);
        const adTitle = $('body > main > div.row.justify-content-center > article >
div:nth-child(2) > header > h1').text().trim();
        const FullAdDescription = $('#dbox > p').text().trim();
        const adDescription = FullAdDescription.replace(/CONTINUE LENDO/g,
'');
        const contactInfoContent =
extractContactInfoFromDescription(adDescription);
        const contactInfo = contactInfoContent.length === 0 ? adLink :
contactInfoContent
        const adPrice = $('body > main > div.row.justify-content-center > article >
div:nth-child(2) > div:nth-child(2) > div >
div.card.border-secondary.rounded-4.shadow.mb-4.p-3 > h3').text().trim();
        const neighborhoodContent = $('body > main >
div.row.justify-content-center > article > div:nth-child(2) > header > h2 >
a.lead.link-dark.align-middle').text();
        // Divide a string de entrada com base na vírgula
        const parts = neighborhoodContent.split(',');
        let neighborhood;
        // Extrai o bairro (assumindo que seja a primeira parte)
        if (parts.length >= 1) {
```

```
                neighborhood = parts[0].trim();
            }

            // Encontra e salva o carrossel de imagens
            const imageCarrousel = [];
            $('.carousel-item').each((index, element) => {
                const source = $(element).find('source');
                const dataSrcset = source.attr('data-srcset');
                if (dataSrcset) {
                    imageCarrousel.push(dataSrcset);
                }
            });

            // Itera através do imageCarrousel e extrai links de imagem
            const imageLinks = [];
            imageCarrousel.forEach((srcset) => {
                const regex = /([^\s,]+)/g;
                const matches = srcset.match(regex);
                if (matches && matches.length > 0) {
                    imageLinks.push(matches[0]);
                }
            });

            const adDetail = {
                title: adTitle,
                description: adDescription,
                price: adPrice,
                link: adLink,
                neighborhood: neighborhood,
                imageLinks: imageLinks,
                contactInfo: contactInfo.length === 0 ? adLink : contactInfo
            };

            return adDetail;
        }
    } catch (error) {
        console.error('Erro durante o scraping de detalhes do anúncio:',
error.message);
    }
  });

  const adDetails = await Promise.all(adDetailPromises);

  return adDetails.filter(Boolean);
}

module.exports = {
  extractMgfHrefValues
}
```

CAMINHO: server/scrapers/imageScraper.js
ARQUIVO: imageScraper.js

```
const https = require('https');
const cheerio = require('cheerio');
const imageUrlPattern =
/https:\/\/cdn\.vistahost\.com\.br\/ibagyimo\/vista\.imobi\/fotos\//;

// Função para fazer scraping de imagens do Classificados UFSC
async function scrapeImagesClassificadosUfsc(imageUrls) {
    return new Promise((resolve, reject) => {
        https.get(imageUrls, (response) => {
            let data = '';

            response.on('data', (chunk) => {
                data += chunk;
            });

            response.on('end', () => {
                const $ = cheerio.load(data);

                const images = [];

                $('img').each((index, element) => {
                    images.push($(element).attr('src'));
                });

                const adImages = images.filter(image => image.includes('images') &&
image.includes('_tmb1.jpg'))
                    .map(image => image.replace('images/',
'https://classificados.inf.ufsc.br/images/')
                        .replace('_tmb1.jpg', '.jpg'));

                resolve(adImages);
            });
        }).on('error', (error) => {
            reject(error);
        });
    });
}

// Função para fazer scraping de imagens do Ibagy
async function scrapeImagesIbagy(url) {
    return new Promise((resolve, reject) => {
        https.get(url, (response) => {
            let data = '';

            response.on('data', (chunk) => {
                data += chunk;
            });
```

```
            response.on('end', () => {
                const $ = cheerio.load(data);

                const images = new Set(); // Usa um Set para armazenar URLs de
imagens únicas

                $('img').each((index, element) => {
                    const imageUrl = $(element).attr('src');
                    if (imageUrl && imageUrlPattern.test(imageUrl)) {
                        images.add(imageUrl);
                    }
                });

                // Converte o Set de volta para um array
                const uniqueImages = Array.from(images);
                resolve(uniqueImages);
            });

            response.on('error', (error) => {
                reject(error);
            });
        });
    });
}

module.exports = {
    scrapeImagesClassificadosUfsc,
    scrapeImagesIbagy
};
```

CAMINHO: server/scrapers/ibagyScraper.js
ARQUIVO: ibagyScraper.js

```
const axios = require('axios');
const axiosRateLimit = require('axios-rate-limit');
const cheerio = require('cheerio');
const { scrapeImagesIbagy } = require('./imageScraper');
const { extractIdfromAdLink, extractPhoneFromWhatsAppLink } =
require('./contactInfoScraper');

const rateLimitedAxios = axiosRateLimit(axios.create(), {
    maxRequests: 2, // Ajuste esse valor com base na política de limitação de
requisições do site
    perMilliseconds: 1000, // Ajuste esse valor com base na política de limitação de
requisições do site
});
```

```
// Função para fazer scraping dos anúncios do Ibagy
async function scrapeIbagyAds() {
    const pages = [
        'https://ibagy.com.br/aluguel/residencial/florianopolis/',
        'https://ibagy.com.br/aluguel/kitnet_conjugado/florianopolis'
    ];

    try {
        const pagePromises = pages.map(async (page) => {
            try {
                const response = await rateLimitedAxios.get(page);

                if (response.status === 200) {
                    const $ = cheerio.load(response.data);
                    const adsPage = $('#imovel-boxes');
                    const adsLinks = new Set();

                    adsPage.find('a[target="_blank"]').each((index, element) => {
                        const link = $(element).attr('href');
                        if (link) {
                            adsLinks.add(link);
                        }
                    });

                    const uniqueAdsLinks = Array.from(adsLinks);
                    return uniqueAdsLinks;
                } else {
                    console.error('Erro ao realizar a busca da página. Código do status:',
response.status);
                    return [];
                }
            } catch (error) {
                console.error('Erro ao fazer o scraping de informações da página:',
error.message);
                return [];
            }
        });

        const allPageLinks = await Promise.all(pagePromises);
        const adLinks = [].concat(...allPageLinks);

        const adItems = await scrapeIbagyAdsDetails(adLinks);
        return adItems;
    } catch (error) {
        console.error('Error:', error.message);
    }
}

// Função para fazer scraping dos detalhes dos anúncios do Ibagy
async function scrapeIbagyAdsDetails(adLinks) {
```

```
    try {
        const adDetailsArray = [];

        const adPromises = adLinks.map(async (link) => {
            const response = await axios.get(link);
            if (response.status === 200) {
                const $ = cheerio.load(response.data);
                const adDescriptionMatch = $('#clb-descricao > div > div > div:nth-child(3)
> p').text();
                const titleMatch = $('#clb-descricao h2').text();
                const title = titleMatch || 'Title not found';
                const price = $('#clb-imovel-topo > div > div:nth-child(1) > div:nth-child(2) >
div.property-thumb-info-label > span > span.thumb-price').text() + ' + taxas';
                const address = $('#section-map > div > div > div > div > p > span').text();
                const neighborhood = extractNeighborhood(address);
                const adIdNumber = extractIdfromAdLink(link);
                const contactLinkObject = $(`#imovelView_asyncSubmit >
div.mauticform-innerform > div > div.propertyform-bottom >
a.clb-gtm-site-whatsapp.clb-gtm-imovel-form-whatsapp.clb-gtm-imovel-${adIdNumbe
r}.clb-interesse-aluguel`);
                let contactInfo =
extractPhoneFromWhatsAppLink(contactLinkObject["0"].attribs.href);
                contactInfo = contactInfo.length === 0 ? link : contactInfo;
                const imageLinks = await scrapeImagesIbagy(link);
                const adDetails = {
                    title,
                    adDescription: adDescriptionMatch,
                    imageLinks,
                    link,
                    price,
                    neighborhood,
                    contactInfo
                };
                adDetailsArray.push(adDetails);
            } else {
                console.error('Erro ao buscar detalhes de anúncios de:', link);
            }
        });

        await Promise.all(adPromises);
        return adDetailsArray;
    } catch (error) {
        console.error('Error:', error.message);
    }
}

// Função para extrair o bairro do endereço
function extractNeighborhood(address) {
    const neighborhoodMatch = /(\d+,\s*)(.*?)\s*-\s*Florianópolis\/Sc/i.exec(address);
    if (neighborhoodMatch) {
```

```
        return neighborhoodMatch[2].trim();
    } else {
        return 'Bairro não encontrado';
    }
}

module.exports = {
    scrapeIbagyAds
};
```

CAMINHO: server/scrapers/contactInfoScraper.js
ARQUIVO: contactInfoScraper.js

```
// Função para extrair informações de contato a partir da descrição do anúncio
function extractContactInfoFromDescription(description) {
    // Padrões de expressões regulares para números de telefone
    const phonePatterns = [
        /\+\d{2}\(\d{2}\)\d{5}-\d{4}/, // +xx(xx)xxxxx-xxxx
        /\+\d{2} \d{2} \d{9}/,          // +xx xx xxxxxxxxx
        /\+\d{2} \d{2} \d{4}-\d{4}/,    // +xx xx xxxx-xxxx
        /\(\+\d{2}\) \d{2} \d{9}/,      // (+xx) xx xxxxxxxxx
        /\(\d{2}\)\d{9}/,               // (xx)xxxxxxxxx
        /\(\d{2}\) \d{5}-\d{4}/,        // (xx) xxxxx-xxxx
        /\d{2} \d{5}-\d{4}/,            // xx xxxxx-xxxx
        /\d{2} \d{8}/,                  // xx xxxxxxxx
        /\d{2}-\d{9}/,                  // xx-xxxxxxxxx
        /\d{2} \d \d{4} \d{4}/,         // xx x xxxx xxxx
        /\d{10}/,                       // xxxxxxxxxx
        /\d{2}\.\d\.\d{4}\.\d{4}/,      // xx.x.xxxx.xxxx
        /\(\d{2}\) \d{9}/,              // (xx) xxxxxxxxx
        /\d{2} \d{3} \d{3} \d{3}/       // xx xxx xxx xxx
    ];

    const uniqueContactInfo = new Set(); // Usar um Set para armazenar números de telefone únicos porque evita duplicatas

    phonePatterns.forEach(pattern => {
        const matches = description.match(pattern);
        if (matches) {
            // Como queremos salvar apenas correspondências únicas, adicionamos a primeira correspondência encontrada
            uniqueContactInfo.add(matches[0]);
        }
    });

    // converte o Set de volta para um array antes de retornar
    return Array.from(uniqueContactInfo);
}
```

```
// Função para extrair o ID do link do anúncio
function extractIdfromAdLink(adLink) {
    // dividir o adLink por "/" para extrair partes individuais
    const parts = adLink.split("/");

    // laço para percorrer as partes para encontrar uma parte que se assemelhe a um
ID
    for (const part of parts) {
        // Verifique se a parte é um ID numérico (assumindo que os IDs são numéricos)
        if (/^\d+$/.test(part)) {
            return part; // Retorna a primeira parte numérica encontrada como o ID
        }
    }

    // Se nenhum ID for encontrado, retorne null ou um valor apropriado
    return null;
}

// Função para extrair o número de telefone de um link do WhatsApp
function extractPhoneFromWhatsAppLink(link) {
    // Defina uma expressão regular para extrair o número de telefone de um link da
API do WhatsApp
    const regex = /(?:\?|&)phone=(\d+)/;

    // Use a expressão regular para encontrar uma correspondência no link
    const match = link.match(regex);

    // Se houver uma correspondência, retorne o número de telefone extraído, caso
contrário, retorne null
    if (match && match[1]) {
        return match[1];
    } else {
        return null;
    }
}

module.exports = {
    extractContactInfoFromDescription,
    extractIdfromAdLink,
    extractPhoneFromWhatsAppLink
};
```

CAMINHO: server/scrapers/classificadosUfscScraper.js
ARQUIVO: classificadosUfscScraper.js

```
const axios = require('axios');
const cheerio = require('cheerio');
const axiosRetry = require('axios-retry');
const axiosRateLimit = require('axios-rate-limit');
const { scrapeImagesClassificadosUfsc } = require('./imageScraper');
const { extractContactInfoFromDescription } = require('./contactInfoScraper');

// Cria uma instância Axios com limite de taxa e tentativas de repetição
const axiosInstance = axiosRateLimit(axios.create(), {
    maxRequests: 2, // Número de requisições por segundo
    perMilliseconds: 1000, // Milissegundos por requisição (neste caso, 1 requisição
por segundo)
});

axiosRetry(axiosInstance, { retries: 3, retryDelay: axiosRetry.exponentialDelay });

// Função para obter os links dos anúncios a partir de uma URL
async function getAdLinks(url) {
    try {
        const response = await axiosInstance.get(url);
        const html = response.data;
        const $ = cheerio.load(html);

        const items = [];

        $('table tr').each((index, element) => {
            const titleColumn = $(element).find('td:nth-child(2)');
            const linkElement = titleColumn.find('a');
            const link = 'https://classificados.inf.ufsc.br/' + linkElement.attr('href');
            if (isValidLink(link)) {
                items.push({ link });
            }
        });

        return items;
    } catch (error) {
        console.error('Erro ao pegar a lista de anúncios:', error);
        return [];
    }
}

// Função para verificar se um link é válido
function isValidLink(link) {
    const linkPattern = /^https:\/\/classificados\.inf\.ufsc\.br\/detail/;
    return linkPattern.test(link);
}
```

```
// Função para obter detalhes dos anúncios a partir dos links
async function getAdDetails(items) {
    const itemsWithDetails = [];

    for (const item of items) {
        try {
            const response = await axios.get(item.link);

            if (response.status === 200) {
                const html = response.data;
                const $ = cheerio.load(html);

                const table = $('#container > div:nth-child(2) > table > tbody >
tr:nth-child(4) > td:nth-child(1) > form > table');
                const priceRow = table.find('tr:has(td[valign="top"]:contains("Preço"))');
                const priceCell = priceRow.find('td[valign="top"]:contains("Preço") + td');
                const price = priceCell.text().trim();
                const genderRow = table.find('tr:has(td[valign="top"]:contains("Gênero"))');
                const genderCell = genderRow.find('td[valign="top"]:contains("Gênero") +
td');
                const gender = genderCell.text().trim();
                const neighborhoodRow =
table.find('tr:has(td[valign="top"]:contains("Bairro"))');
                const neighborhoodCell =
neighborhoodRow.find('td[valign="top"]:contains("Bairro") + td');
                const neighborhood = neighborhoodCell.text().trim();

                // Verifica se algum dos valores críticos está vazio
                if (!price || !gender || !neighborhood) {
                    console.warn(`Faltaram informações para o anúncio: ${item.link}. Ele
será ignorado`);
                    continue; // Pula este anúncio e passa para o próximo
                }

                const box = $('.box');
                const title = box.find('h1').text().trim();
                let description = box.find('[colspan="2"]').text().trim();

                const regexMatch = description.match(/Descrição:(.*?)Detalhes Gerais:/s);
                if (regexMatch && regexMatch[1]) {
                    description = regexMatch[1].trim();
                }
                const imageUrls = item.link + '&show_still=1';

                const adImages = await scrapeImagesClassificadosUfsc(imageUrls);
                // Extrai informações de contato da descrição
                let contactInfo = extractContactInfoFromDescription(description);
                contactInfo = contactInfo.length === 0 ? item.link : contactInfo;
                itemsWithDetails.push({
                    title: title,
```

```
                link: item.link,
                description: description,
                price: price,
                imageLinks: adImages,
                contactInfo: contactInfo,
                neighborhood: neighborhood
            });
        } else {
            console.error(`Erro ao pegar detalhes de anúncio em ${item.link}: Status
${response.status}`);
        }
    } catch (error) {
        console.error(`Erro ao pegar detalhes de anúncio em ${item.link}: ${error}`);
    }
  }
  return itemsWithDetails;
}

module.exports = {
  getAdLinks,
  getAdDetails
};
```

CAMINHO: server/routes/api/users.js
ARQUIVO: users.js

```
const express = require("express");
const router = express.Router();
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const keys = require("../../config/keys");
const passport = require("passport");
const User = require("../../models/User"); // Carrega o modelo User
const validateRegisterInput = require("../../validation/register"); // Carrega a validação
de entrada para registro
const validateLoginInput = require("../../validation/login");

// Rota POST para registro de usuário
router.post("/register", (req, res) => {
  // Validação do formulário
  const { errors, isValid } = validateRegisterInput(req.body);

  // Verifica a validação
  if (!isValid) {
    return res.status(400).json(errors);
  }

  User.findOne({ email: req.body.email }).then(user => {
```

```
        if (user) {
            return res.status(400).json({ email: "O email já existe" });
        } else {
            const newUser = new User({
                name: req.body.name,
                email: req.body.email,
                password: req.body.password,
                preferences: {
                    houseOrApartment: req.body.preferences.houseOrApartment,
                    genderPreference: req.body.preferences.genderPreference,
                    acceptsPets: req.body.preferences.acceptsPets,
                    location: req.body.preferences.location,
                    roommates: req.body.preferences.roommates,
                    leaseLength: req.body.preferences.leaseLength,
                    budget: req.body.preferences.budget,
                    wheelchairAccessible: req.body.preferences.wheelchairAccessible,
                    noiseLevel: req.body.preferences.noiseLevel,
                    acceptSmoker: req.body.preferences.acceptSmoker,
                    hasFurniture: req.body.preferences.hasFurniture
                }
            });

            // Hash da senha antes de salvar no banco de dados
            bcrypt.genSalt(10, (err, salt) => {
                bcrypt.hash(newUser.password, salt, (err, hash) => {
                    if (err) throw err;
                    newUser.password = hash;
                    newUser
                        .save()
                        .then(user => res.json(user))
                        .catch(err => console.log(err));
                });
            });
        }
    });
});

// Rota POST para login de usuário e retorno do token JWT
router.post("/login", (req, res) => {
    // Validação do formulário
    const { errors, isValid } = validateLoginInput(req.body);

    // Verifica a validação
    if (!isValid) {
        return res.status(400).json(errors);
    }

    const email = req.body.email;
    const password = req.body.password;
```

```
    // Encontrar usuário por email
    User.findOne({ email }).then(user => {
        // Verifica se o usuário existe
        if (!user) {
            return res.status(404).json({ emailnotfound: "O email não foi encontrado" });
        }

        // Verifica a senha
        bcrypt.compare(password, user.password).then(isMatch => {
            if (isMatch) {
                // Usuário correspondido
                // Criação do Payload JWT
                const payload = {
                    id: user.id,
                    name: user.name
                };

                // Assina o token
                jwt.sign(
                    payload,
                    keys.secretOrKey,
                    {
                        expiresIn: 31556926 // 1 ano em segundos
                    },
                    (err, token) => {
                        res.json({
                            success: true,
                            token: "Bearer " + token
                        });
                    }
                );
            } else {
                return res
                    .status(400)
                    .json({ passwordincorrect: "A senha está incorreta" });
            }
        });
    });
});

// Rota PUT para atualizar as preferências do usuário
router.put("/preferences", passport.authenticate("jwt", { session: false }), (req, res) =>
{
    // O usuário está autenticado, prossegue com a atualização de preferências
    const updatedPreferences = req.body;

    // Atualiza as preferências do usuário no banco de dados
    User.findOneAndUpdate(
        { _id: req.user.id }, // ID do usuário proveniente do payload do JWT
        { $set: { "preferences": updatedPreferences } }, // Especifica o campo
```

```
"preferences"
        { new: true }
    )
        .then((updatedUser) => {
            if (!updatedUser) {
                return res.status(404).json({ userNotFound: "Usuário não encontrado" });
            }
            res.json(updatedUser.preferences); // Responde com as preferências
atualizadas
        })
        .catch((err) => {
            console.log(err);
            return res.status(500).json({ error: "Erro do Servidor" });
        });
});

// Rota GET para obter as preferências do usuário
router.get("/preferences", passport.authenticate("jwt", { session: false }), (req, res) =>
{
    // O usuário está autenticado, recupera suas preferências
    User.findById(req.user.id)
        .then((user) => {
            if (!user) {
                return res.status(404).json({ userNotFound: "Usuário não encontrado" });
            }
            res.json(user.preferences); // Responde com as preferências do usuário
        })
        .catch((err) => {
            console.log(err);
            return res.status(500).json({ error: "Erro do Servidor" });
        });
});

// Rota GET para obter as informações do usuário atual
router.get("/me", passport.authenticate("jwt", { session: false }), (req, res) => {
    // O usuário está autenticado, recupera suas informações
    User.findById(req.user.id)
        .then((user) => {
            if (!user) {
                return res.status(404).json({ userNotFound: "Usuário não encontrado" });
            }
            res.json(user); // Responde com as informações do usuário
        })
        .catch((err) => {
            console.log(err);
            return res.status(500).json({ error: "Erro do Servidor" });
        });
});

// Rota DELETE para excluir a conta do usuário
```

```
router.delete("/delete", passport.authenticate("jwt", { session: false }), (req, res) => {
    // O usuário está autenticado, prossegue com a exclusão da conta
    User.findByIdAndRemove(req.user.id)
        .then(() => {
            res.json({ success: true, message: "Conta de usuário excluída com sucesso"
});
        })
        .catch((err) => {
            console.log(err);
            return res.status(500).json({ error: "Erro do Servidor" });
        });
});

module.exports = router;
```

CAMINHO: server/routes/api/recommendation.js
ARQUIVO: recommendation.js

```
const express = require('express');
const router = express.Router();
const passport = require("passport");
const User = require("../../models/User"); // Carrega o modelo User
const generateRecommendations =
require('../../recommendations/adRecommendation'); // Importa a função para gerar
recomendações com base nas preferências do usuário

// Rota para obter recomendações baseadas em conteúdo para um usuário
router.get('/all', passport.authenticate("jwt", { session: false }), async (req, res) => {
    try {
        const userId = req.user.id;
        // Busca as preferências do usuário
        const user = await User.findById(userId);
        const userPreferences = user.preferences;
        // Gera recomendações com base nas preferências do usuário
        const recommendations = await generateRecommendations(userPreferences);

        res.json(recommendations);
    } catch (error) {
        console.error("Erro ao gerar os anúncios recomendados baseados em
conteúdo:", error);
        res.status(500).json({ message: 'Erro ao listar as recomendações', error:
error.message });
    }
});

module.exports = router;
```

CAMINHO: server/routes/api/ads.js
ARQUIVO: ads.js

```
const express = require('express');
const router = express.Router();
const adController = require('../../controller/adsController'); // Importa o controlador

// Rota para obter todos os anúncios
router.get('/all', adController.getAllAds);

// Rota para obter LAST_SCRAPING_DATE
router.get('/lastScrapingDate', adController.getLastScrapingDate);

module.exports = router;
```

CAMINHO: server/recommendations/adRecommendation.js
ARQUIVO: adRecommendation.js

```
const Ad = require('../models/Ads');

// Função para buscar anúncios no banco de dados e gerar recomendações
async function generateRecommendations(userPreferences) {
    try {
        // Busca todos os anúncios no banco de dados
        const ads = await Ad.find({});
        let sampleSize; // Tamanho padrão da amostra

        const totalAds = ads.length;

        // Ajusta o tamanho da amostra com base em vários fatores
        if (totalAds <= 50) {
            sampleSize = totalAds; // Usa todos os anúncios se houver muito poucos.
        } else if (totalAds <= 500) {
            sampleSize = 100; // Limita o tamanho da amostra se houver entre 6 e 10
anúncios.
        } else if (totalAds <= 1000) {
            sampleSize = 250; // Um tamanho de amostra maior para conteúdo mais
diversificado.
        } else {
            sampleSize = 300; // Para bancos de dados maiores, um tamanho de
amostra razoável.
        }
        // Inicializa uma matriz para armazenar anúncios recomendados
        const recommendations = [];

        // Analisa a preferência de orçamento do usuário
        const userBudget = parseFloat(userPreferences.budget);
```

```
      // Percorre os anúncios e calcula uma pontuação para cada anúncio com base
nas preferências do usuário
       ads.forEach((ad) => {
          // Extrai informações de orçamento da descrição do anúncio e converte para
um valor numérico
          const adBudgetMatch =
ad.description.match(/\bR\$\s?\d{3,}(?:,\d{1,2})?|\$\s?\d{3,}(?:,\d{1,2})?|\d{3,}(?:,\d{1,
2})?\b/);
          const adBudget = adBudgetMatch ?
parseFloat(adBudgetMatch[0].replace(/[^\d.]/g, '')) : null;

          if (adBudget === null || adBudget <= userBudget) {
             // Extrai outras características do anúncio
             const adFeatures = {
                houseOrApartment: ad.description.match(/casa|apartamento/i),
                genderPreference:
ad.description.match(/homem|mulher|masculino|feminino|masculina|feminina/i),
                acceptsPets: ad.description.match(/aceita pets|pets permitidos/i),
                location: ad.neighborhood,
                roommates: ad.description.match(/alugo quarto|aluga-se quarto|quarto
disponível|quarto compartilhado/i),
                leaseLength: ad.description.match(/aluguel anual|aluguel mensal|alugo
mensal|alugo anual|aluguel temporada/i),
                wheelchairAccessible: ad.description.match(/acessível a
cadeirantes|acesso à cadeirantes|acesso à cadeira de rodas/i),
                noiseLevel: ad.description.match(/tranquilo|barulhento|local
tranquilo|local perto do centro/i),
                acceptSmoker: ad.description.match(/aceita fumante|fumante
permitido/i),
                hasFurniture: ad.description.match(/mobiliado|tem mobilia|possui
móveis|possui moveis|tem algumas mobilias|mobilia inclusa/i)
             };

             // Calcula uma pontuação para o anúncio com base nas preferências do
usuário
             let score = 0;

             for (const feature in adFeatures) {
                if (userPreferences[feature] && adFeatures[feature]) {
                   score++;
                }
             }

             // Adiciona o anúncio e sua pontuação às recomendações
             recommendations.push({
                ad,
                score,
             });
          }
       });
```

```
        // Ordena as recomendações por pontuação em ordem decrescente
        recommendations.sort((a, b) => b.score - a.score);

        // Retorna os anúncios recomendados como um objeto JSON
        return recommendations.slice(0, sampleSize);
    } catch (error) {
        console.error('Erro ao listar anúncios ou gerar recomendações:', error);
        return [];
    }
}
module.exports = generateRecommendations;
```

CAMINHO: server/models/Ads.js
ARQUIVO: Ads.js

```
const mongoose = require('mongoose');

// Definição do esquema para anúncios
const adSchema = new mongoose.Schema({
    title: String, // Título do anúncio
    link: String, // Link relacionado ao anúncio
    description: String, // Descrição do anúncio
    price: String, // Preço do anúncio
    imageLinks: Array, // Array de links para imagens relacionadas ao anúncio
    neighborhood: String, // Bairro do anúncio
    contactInfo: Array, // Array de informações de contato relacionadas ao anúncio
    source: String // Fonte do anúncio
});

// Modelo para a coleção de anúncios
const Ad = mongoose.model('Ad', adSchema);

// Exporta o modelo de anúncio
module.exports = Ad;
```

CAMINHO: server/models/User.js
ARQUIVO: User.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

// Definição do esquema de preferências do usuário
const PreferencesSchema = new Schema({
    houseOrApartment: {
        type: String,
        enum: ["casa", "apartamento"] // Opções válidas: casa ou apartamento
    },
    genderPreference: {
```

```
        type: String,
        enum: ["masculino", "feminino", "tanto faz", "ambos"] // Opções válidas:
masculino, feminino, tanto faz, ambos
    },
    acceptsPets: {
        type: Boolean // Aceita animais de estimação (true/false)
    },
    location: {
        type: String // Localização preferida
    },
    roommates: {
        type: String,
        enum: ["sozinho", "compartilhado"] // Opções válidas: sozinho ou compartilhado
    },
    leaseLength: {
        type: String,
        enum: ["aluguel anual", "aluguel temporada"] // Opções válidas: aluguel anual
ou aluguel temporada
    },
    budget: {
        type: String // Orçamento desejado
    },
    wheelchairAccessible: {
        type: Boolean // Acessível a cadeirantes (true/false)
    },
    noiseLevel: {
        type: String,
        enum: ["tranquilo", "barulhento"] // Opções válidas: tranquilo ou barulhento
    },
    acceptSmoker: {
        type: Boolean // Aceita fumante (true/false)
    },
    hasFurniture: {
        type: Boolean // Possui mobília (true/false)
    },
});

// Definição do esquema de usuário, que inclui as preferências
const UserSchema = new Schema({
    name: {
        type: String,
        required: true // Nome do usuário (obrigatório)
    },
    email: {
        type: String,
        required: true // Endereço de e-mail do usuário (obrigatório)
    },
    password: {
        type: String,
        required: true // Senha do usuário (obrigatório)
```

```
    },
    date: {
        type: Date,
        default: Date.now // Data de criação do usuário (padrão: data atual)
    },
    preferences: PreferencesSchema // Incorpora o esquema de preferências ao
esquema de usuário
});

// Exporta o modelo de usuário
module.exports = User = mongoose.model("users", UserSchema);
```

CAMINHO: server/controller/adsController.js
ARQUIVO: adsController.js

```
const Ad = require("../models/Ads");
const scraperUfsc = require("../scrapers/classificadosUfscScraper");
const { scrapeIbagyAds } = require("../scrapers/ibagyScraper");
const { scrapeWebQuartoads } = require("../scrapers/webQuartoScraper");
const { getVivaRealAdLinks } = require("../scrapers/vivaRealScraper");
const { extractMgfHrefValues } = require("../scrapers/mgfScraper");
const moment = require('moment-timezone');

// Define um array de URLs
const urls = [
    {
        "url": "https://classificados.ufsc.br/index.php?catid=88"
    },
    {
        "url": "https://classificados.ufsc.br/index.php?catid=91"
    },
    {
        "url": "https://classificados.ufsc.br/index.php?catid=89"
    },
    {
        "url": "https://classificados.ufsc.br/index.php?catid=94"
    },
    {
        "url": "https://classificados.ufsc.br/index.php?catid=197"
    },
    {
        "url": "https://classificados.ufsc.br/index.php?catid=90"
    },
    {
        "url": "https://classificados.ufsc.br/index.php?catid=96"
    },
    {
        "url": "https://classificados.ufsc.br/index.php?catid=86"
```

```
    },
    {
        "url": "https://classificados.ufsc.br/index.php?catid=72"
    },
    {
        "url": "https://classificados.ufsc.br/index.php?catid=73"
    },
    {
        "url": "https://classificados.ufsc.br/index.php?catid=74"
    }
];

// Função para raspar e salvar novos anúncios, evitando duplicatas
async function scrapeAndSaveNewAds(scraper, source, urls, scrapeFunction,
getDetailsFunction) {
    try {
        if (urls) {
            for (const urlInfo of urls) {
                // Busca anúncios usando a função de raspagem fornecida
                const adItems = await scrapeFunction(urlInfo.url);

                // Busca anúncios existentes no banco de dados
                const existingAds = await Ad.find({}, "link");

                // Filtra novos anúncios
                const newAdItems = adItems.filter(
                    (item) =>
                        !existingAds.some((existingAd) => existingAd.link === item.link)
                );

                if (newAdItems.length > 0) {
                    // Busca detalhes do anúncio usando a função getDetailsFunction, se
disponível
                    const itemsWithDetails = getDetailsFunction
                        ? await getDetailsFunction(newAdItems)
                        : newAdItems;

                    // Mapeia e salva novos anúncios
                    const finalItems = itemsWithDetails.map((item) => ({
                        title: item.title,
                        link: item.link,
                        description: item.description,
                        price: item.price,
                        imageLinks: item.imageLinks,
                        neighborhood: item.neighborhood,
                        contactInfo: item.contactInfo,
                    }));

                    await saveNewAds(finalItems, source);
                } else {
```

```
                console.log(`Sem novos anúncios de ${source} para salvar.`);
            }
        }
    } else {
        // Busca anúncios usando a função de raspagem fornecida
        const adItems = await scrapeFunction();

        // Busca anúncios existentes no banco de dados
        const existingAds = await Ad.find({}, "link");

        // Filtra novos anúncios
        const newAdItems = adItems.filter(
            (item) =>
                !existingAds.some((existingAd) => existingAd.link === item.link)
        );

        if (newAdItems.length > 0) {
            // Mapeia e salva novos anúncios
            const finalItems = newAdItems.map((item) => ({
                title: item.title,
                link: item.link || "",
                description: item.description || "",
                price: item.price || "",
                imageLinks: item.imageLinks || "",
                neighborhood: item.neighborhood,
                contactInfo: item.contactInfo,
                source: source,
            }));

            await saveNewAds(finalItems, source);
        } else {
            console.log(`Sem novos anúncios de ${source} para salvar.`);
        }
    }
} catch (error) {
    console.error(`Erro durante o scraping de anúncios de ${source}:`, error);
}
}

// Função para salvar novos anúncios no banco de dados, evitando duplicatas
async function saveNewAds(newAds, source) {
    try {
        const existingAds = await Ad.find({}, "link");
        const newAdsToInsert = newAds.filter(
            (newAd) => !existingAds.some((existingAd) => existingAd.link ===
newAd.link)
        );

        if (newAdsToInsert.length > 0) {
            const finalAds = newAdsToInsert.map((item) => ({
```

```
                title: item.title,
                link: item.link || "",
                description: item.description || "",
                price: item.price || "",
                imageLinks: item.imageLinks || "",
                neighborhood: item.neighborhood,
                contactInfo: item.contactInfo,
                source: source,
            }));

            await Ad.insertMany(finalAds);
        } else {
            console.log(`Sem novos anúncios de ${source} para salvar.`);
        }
    } catch (error) {
        console.error(`Erro durante o scraping de anúncios de ${source}:`, error);
    }
}

// Função para iniciar o processo de raspagem
async function startScraping() {
    try {
        console.log("O processo de Scraping começou.");

        await scrapeAndSaveNewAds(
            scraperUfsc,
            "Classificados UFSC",
            urls,
            scraperUfsc.getAdLinks,
            scraperUfsc.getAdDetails
        );
        console.log("O processo de Scraping para Classificados UFSC foi finalizado.");

        await scrapeAndSaveNewAds(
            null,
            "Ibagy",
            null,
            scrapeIbagyAds,
            null
        );
        console.log("O processo de Scraping para Ibagy foi finalizado.");

        await scrapeAndSaveNewAds(
            null,
            "WebQuarto",
            null,
            scrapeWebQuartoads,
            null
        );
        console.log("O processo de Scraping para WebQuarto foi finalizado.");
```

```
        await scrapeAndSaveNewAds(
            null,
            "vivaReal",
            null,
            getVivaRealAdLinks,
            null
        );
        console.log("O processo de Scraping para VivaReal foi finalizado.");

        await scrapeAndSaveNewAds(
            null,
            "MGF",
            null,
            extractMgfHrefValues,
            null
        );
        console.log("O processo de Scraping para MGF foi finalizado.");

        console.log("Todos os processos de scraping foram finalizados.");
        const now = moment().tz("America/Sao_Paulo").format(); // Obtém a data atual
no formato UTC-03:00
        process.env.LAST_SCRAPING_DATE = now;
        console.log(now);
        console.log("LAST_SCRAPING_DATE foi atualizado:",
process.env.LAST_SCRAPING_DATE);
    } catch (error) {
        console.error("Erro durante o scraping:", error);
    }
}

// Função para buscar todos os anúncios
const getAllAds = async (req, res) => {
    try {
        const ads = await Ad.find();
        res.json(ads);
    } catch (error) {
        res.status(500).json({ message: 'Erro ao listar anúncios', error: error.message
});
    }
};

// Função para buscar a LAST_SCRAPING_DATE
const getLastScrapingDate = (req, res) => {
    try {
        const lastScrapingDate = process.env.LAST_SCRAPING_DATE;
        if (lastScrapingDate) {
            res.json({ lastScrapingDate });
        } else {
            res.status(404).json({ message: 'LAST_SCRAPING_DATE não encontrado'
```

```
});
        }
    } catch (error) {
        res.status(500).json({ message: 'Erro ao buscar LAST_SCRAPING_DATE',
error: error.message });
    }
};

module.exports = {
    startScraping,
    getAllAds,
    getLastScrapingDate,
};
```

CAMINHO: server/config/keys.js
ARQUIVO: keys.js

```
// Exporta a chave secreta usada para assinar e verificar tokens JWT
module.exports = {
    secretOrKey: "secret" // Chave secreta utilizada para tokens JWT
};
```

CAMINHO: server/config/passport.js
ARQUIVO: passport.js

```
// Exporta a chave secreta usada para assinar e verificar tokens JWT
module.exports = {
    secretOrKey: "secret" // Chave secreta utilizada para tokens JWT
};
```

CAMINHO: client/public/index.html
ARQUIVO: index.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="utf-8">
  <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
  <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
  <meta name="theme-color" content="#000000">
  <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
  <!-- CSS compilado e minificado -->
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css"
>
```

```
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
  <title>VagouAqui</title>
</head>
<body>
<noscript>
  É necessário habilitar o JavaScript para executar este aplicativo.
</noscript>
<div id="root"></div>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></s
cript>
</body>
</html>
```

CAMINHO: client/src/App.js
ARQUIVO: App.js

```
import React, { Component } from "react";
import { BrowserRouter as Router, Route, Switch } from "react-router-dom";
import jwt_decode from "jwt-decode";
import setAuthToken from "./utils/setAuthToken";
import { setCurrentUser, logoutUser } from "./actions/authActions.js";
import { Provider } from "react-redux";
import store from "./store";
import Navbar from "./views/layout/Navbar";
import Landing from "./views/layout/MainPage";
import Register from "./views/User/Register";
import Login from "./views/User/Login";
import PrivateRoute from "./utils/PrivateRoute";
import Dashboard from "./views/layout/Landing";
import AdGrid from "./views/adGrid/AdGrid";
import EditPreferences from "./views/User/EditPreferences";
import RecommendedGrid from "./views/adGrid/RecommendedGrid";
import UserPage from "./views/User/UserPage";

// Verifica se há um token para manter o usuário logado
if (localStorage.jwtToken) {
    // Define o cabeçalho do token do usuário
    const token = localStorage.jwtToken;
    setAuthToken(token);
    // Decodifica o token e obtém as informações do usuário e expiração
    const decoded = jwt_decode(token);
    // Define o usuário e isAuthenticated
    store.dispatch(setCurrentUser(decoded));

    // Verifica se o token expirou
```

```
    const currentTime = Date.now() / 1000; // para obter em milissegundos
    if (decoded.exp < currentTime) {
        // Faz logout do usuário
        store.dispatch(logoutUser());
        // Redireciona para a página de login
        window.location.href = "./login";
    }
}

class App extends Component {
    render() {
        return (
            <Provider store={store}>
                <Router>
                    <div className="App">
                        <Navbar />
                        <Route exact path="/" component={Landing} />
                        <Route exact path="/register" component={Register} />
                        <Route exact path="/login" component={Login} />
                        <Switch>
                            <PrivateRoute exact path="/dashboard" component={Dashboard}
/>
                            <PrivateRoute exact path="/products" component={AdGrid} />
                            <PrivateRoute exact path="/preferences"
component={EditPreferences} />
                            <PrivateRoute exact path="/recommendation"
component={RecommendedGrid} />
                            <PrivateRoute exact path="/me" component={UserPage} />
                        </Switch>
                    </div>
                </Router>
            </Provider>
        );
    }
}
export default App;
```

CAMINHO: client/src/index.js
ARQUIVO: index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

// Renderiza o componente principal (App) no elemento com id 'root'
ReactDOM.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>,
```

```
    document.getElementById('root')
);
```

CAMINHO: client/src/store.js
ARQUIVO: store.js

```
import { createStore, applyMiddleware, compose } from "redux";
import thunk from "redux-thunk";
import rootReducer from "./reducers";

// Estado inicial do Redux
const initialState = {};

// Middleware utilizado (redux-thunk para ações assíncronas)
const middleware = [thunk];

// Configuração da loja Redux (store)
const store = createStore(
    rootReducer, // Reducer combinado
    initialState, // Estado inicial
    compose(
        applyMiddleware(...middleware), // Aplica middleware
        // Extensão para o navegador Redux DevTools (se disponível)
        window.__REDUX_DEVTOOLS_EXTENSION__ ?
window.__REDUX_DEVTOOLS_EXTENSION__() : (f) => f
    )
);

export default store;
```

CAMINHO: client/src/views/User/UserPage.js
ARQUIVO: UserPage.js

```
import React, { Component } from "react";
import axios from "axios";
import "./UserPage.css";
import { withRouter } from "react-router-dom";
import { connect } from "react-redux";
import PropTypes from "prop-types";
import { logoutUser } from "../../actions/authActions"; // Importa a ação logoutUser

class UserPage extends Component {
    constructor() {
        super();
        this.state = {
            user: null,
            error: null,
```

```
        confirmDelete: false,
    };
}

componentDidMount() {
    this.fetchUserData();
}

fetchUserData = () => {
    axios
        .get("/api/users/me")
        .then((response) => {
            this.setState({ user: response.data });
        })
        .catch((error) => {
            this.setState({ error: "Erro ao obter informações do usuário" });
        });
};

handleDeleteAccount = () => {
    // Exibe a caixa de diálogo de confirmação
    this.setState({ confirmDelete: true });
};

confirmDelete = () => {
    // Usuário confirmou a exclusão da conta
    axios
        .delete("/api/users/delete")
        .then(() => {
            // Despacha a ação logoutUser para desconectar o usuário
            this.props.logoutUser();
            // Redireciona para a página inicial após a exclusão bem-sucedida
            this.props.history.push("/");
        })
        .catch((error) => {
            console.log(error);
            // Trata erro de exclusão
        });
};

cancelDelete = () => {
    // Usuário cancelou a exclusão da conta
    this.setState({ confirmDelete: false });
};

render() {
    const { user, error, confirmDelete } = this.state;

    return (
        <div className="user-profile">
```

```
            {error && <p className="error">{error}</p>}
            {user && (
               <div className="user-profile-info">
                  <div>
                  <h2>Perfil</h2>
                  <p className="user-profile-label"><b>Nome:</b></p><b
className="user-profile-value">{user.name}</b>
                  <p className="user-profile-label"><b>Email:</b></p><b
className="user-profile-value">{user.email}</b>
                  <p className="user-profile-label"><b>Prefiro casa ou
apartamento:</b></p><b
className="user-profile-value">{user.preferences.houseOrApartment}</b>
                  <p className="user-profile-label"><b>Anúncios com colegas de
quarto do gênero:</b></p><b
className="user-profile-value">{user.preferences.genderPreference}</b>
                  <p className="user-profile-label"><b>Anúncios com
pets:</b></p><b className="user-profile-value">{user.preferences.acceptsPets ?
'Sim' : 'Não'}</b>
                  <p className="user-profile-label"><b>Preferência por anúncios
localizados em:</b></p><b
className="user-profile-value">{user.preferences.location}</b>
                  <p className="user-profile-label"><b>Preferência por anúncios de
aluguéis compartilhados:</b></p><b
className="user-profile-value">{user.preferences.roommates}</b>
                  <p className="user-profile-label"><b>Preferência por anúncios do
tipo:</b></p><b
className="user-profile-value">{user.preferences.leaseLength}</b>
                  <p className="user-profile-label"><b>Preferência por anúncios que
aceitem fumante:</b></p><b
className="user-profile-value">{user.preferences.acceptSmoker ? 'Sim' : 'Não'}</b>
                  <p className="user-profile-label"><b>Preferência por anúncios que
possuem mobília:</b></p><b
className="user-profile-value">{user.preferences.hasFurniture ? 'Sim' : 'Não'}</b>
                  <p className="user-profile-label"><b>Preferência por anúncios de
locais com nível de barulho:</b></p><b
className="user-profile-value">{user.preferences.noiseLevel}</b>
                  <p className="user-profile-label"><b>Preferência por anúncios de
locais acessíveis à cadeirantes:</b></p><b
className="user-profile-value">{user.preferences.wheelchairAccessible ? 'Sim' :
'Não'}</b>
               </div>
            {confirmDelete ? (
                     <div>
                        <p className="text-warning">
                           Tem certeza de que deseja excluir a sua conta?
                        </p>
                        <button
                           className="btn btn-danger-delete"
                           onClick={this.confirmDelete}
                        >
```

```
                        Confirmar Exclusão
                    </button>
                    <button
                        className="btn btn-secondary"
                        onClick={this.cancelDelete}
                    >
                        Cancelar
                    </button>
                </div>
            ) : (
                <button
                    className="btn btn-danger-delete"
                    onClick={this.handleDeleteAccount}
                >
                    Deletar conta
                </button>
            )}
        </div>
    )}
  </div>
);
}
}

UserPage.propTypes = {
  auth: PropTypes.object.isRequired,
};

const mapStateToProps = (state) => ({
  auth: state.auth,
});

export default connect(mapStateToProps, { logoutUser })(withRouter(UserPage));
```

CAMINHO: client/src/views/User/UserPage.css
ARQUIVO: UserPage.css

```
.user-profile {
  background-image:
url('https://media.gettyimages.com/id/660340054/pt/foto/space-problem-in-bedroom-i
nterior.jpg?s=612x612&w=0&k=20&c=k8AlqAdhbcVz-hWlmOq49cB5U29P99LJV1H
6A4jlgew=');
  background-size: contain;
  background-repeat: no-repeat;
  align-content: center;
  padding: 20px;
  margin-left: 508px;
  margin-top: 500px;
  max-width: 400px;
```

```
}

.user-profile-info {
    margin-top: -541px;
}

.user-info div {
    margin: 10px 0;
}

.user-profile-label {
    font-size: 20px;
    margin-bottom: 5px;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
}

.user-profile-value {
    font-size: 20px;
    color: #041f3b;
    font-weight: bold;
}

.btn-danger-delete {
    margin-right: 10px;
    padding: 1px;
    font-weight: bold;
}

.btn-secondary {
    font-weight: bold;
}

.text-warning {
    font-weight: bold;
    color: #9d0101;
}

@media screen and (max-width: 576px) {
    .user-profile {
        max-width: 100%;
        background-size: contain;
        background-position: center;
        padding: 10px;
        margin-left: 0;
        margin-top: 0;
    }
    .user-profile-info {
        margin-top: 0;
    }
    .user-info div {
```

```
        margin: 5px 0;
    }
    .user-profile-label {
        font-size: 18px;
    }
    .user-profile-value {
        font-size: 18px;
    }
    .btn-danger-delete {
        margin-right: 5px;
    }
    .btn-secondary {
        margin-right: 5px;
    }
    .text-warning {
        font-size: 18px;
    }
}

@media screen and (min-width: 577px) and (max-width: 992px) {
    .user-profile {
        /* Adjust styles for medium screens */
        max-width: 300px;
    }
}

@media screen and (min-width: 993px) {
    .user-profile {
        max-width: 400px;
    }
}

@media screen and (min-width: 768px) and (max-width: 1024px) {
    .user-profile {
        max-width: 300px;
    }
}
```

CAMINHO:client/src/views/User/Register.js
ARQUIVO: Register.js

```
import React, { Component } from "react";
import { Link, withRouter } from "react-router-dom";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import { registerUser } from "../../actions/authActions.js";
import classnames from "classnames";
import "./Register.css";
```

```
class Register extends Component {
    constructor() {
        super();
        this.state = {
            name: "",
            email: "",
            password: "",
            password2: "",
            errors: {},
            preferences: {
                houseOrApartment: "casa",
                genderPreference: "masculino",
                acceptsPets: false,
                location: "",
                roommates: "sozinho",
                leaseLength: "aluguel anual",
                budget: "",
                wheelchairAccessible: false,
                noiseLevel: "tranquilo",
                acceptSmoker: false,
                hasFurniture: false
            }
        };
    }

    componentDidMount() {
        if (this.props.auth.isAuthenticated) {
            this.props.history.push("/dashboard");
        }
    }

    componentWillReceiveProps(nextProps) {
        if (nextProps.errors) {
            this.setState({
                errors: nextProps.errors
            });
        }
    }

    onChange = e => {
        if (e.target.id.startsWith("preferences.")) {
            // Lidar com mudanças nos campos de preferência
            const preferenceField = e.target.id.split(".")[1];
            const newValue = e.target.type === "checkbox" ? e.target.checked :
e.target.value;

            this.setState(prevState => ({
                preferences: {
                    ...prevState.preferences,
                    [preferenceField]: newValue
```

```
            }
        }));
    } else {
        this.setState({ [e.target.id]: e.target.value });
    }
};

onSubmit = e => {
    e.preventDefault();
    const newUser = {
        name: this.state.name,
        email: this.state.email,
        password: this.state.password,
        password2: this.state.password2,
        preferences: this.state.preferences
    };
    this.props.registerUser(newUser, this.props.history);
};

render() {
    const { errors, preferences } = this.state;

    return (
        <div className="register-container">
            <div className="image-container">
                <div className="row">
                    <div className="form-container">
                        <Link to="/" className="btn-flat-back">
                            <i className="material-icons left">keyboard_backspace</i>
Voltar ao início
                        </Link>
                        <div className="form-container">
                            <h4 className="form-title-register">
                                <b>Cadastre-se</b> abaixo
                            </h4>
                            <p className="form-container-login">
                                Já possui uma conta? <Link to="/login"
className="login-link"><b>Fazer log in</b></Link>
                            </p>
                        </div>
                        <form noValidate onSubmit={this.onSubmit}>
                            <div className="form-field-input">
                                <input
                                    onChange={this.onChange}
                                    value={this.state.name}
                                    error={errors.name}
                                    id="name"
                                    name="name"
                                    type="text"
                                    className={classnames("", {
```

```
                        invalid: errors.name
                    })}
                />
                <label htmlFor="name"
classNameName="form-label">Nome</label>
                <span classNameName="red-text">{errors.name}</span>
            </div>
            <div classNameName="form-field-input">
                <input
                    onChange={this.onChange}
                    value={this.state.email}
                    error={errors.email}
                    id="email"
                    name="email"
                    type="email"
                    classNameName={classnames("", {
                        invalid: errors.email
                    })}
                />
                <label htmlFor="email"
classNameName="form-label">Email</label>
                <span classNameName="red-text">{errors.email}</span>
            </div>
            <div classNameName="form-field-input">
                <input
                    onChange={this.onChange}
                    value={this.state.password}
                    error={errors.password}
                    id="password"
                    name="password"
                    type="password"
                    classNameName={classnames("", {
                        invalid: errors.password
                    })}
                />
                <label htmlFor="password" classNameName="form-label">Digitar
Senha</label>
                <span classNameName="red-text">{errors.password}</span>
            </div>
            <div classNameName="form-field-input">
                <input
                    onChange={this.onChange}
                    value={this.state.password2}
                    error={errors.password2}
                    id="password2"
                    name="password2"
                    type="password"
                    classNameName={classnames("", {
                        invalid: errors.password2
                    })}
```

```
                        />
                        <label htmlFor="password2"
className="form-label">Confirmar Senha</label>
                        <span className="red-text">{errors.password2}</span>
                    </div>

                    {/* Campos de Preferência */}
                    <div className="form-field-input">
                        <input
                            id="preferences.location"
                            name="location"
                            type="text"
                            value={preferences.location}
                            onChange={this.onChange}
                        />
                        <label htmlFor="preferences.location"
className="form-label">Bairro preferencial para locação</label>
                    </div>
                    <div className="form-field-input">
                        <input
                            id="preferences.budget"
                            name="budget"
                            type="text"
                            value={preferences.budget}
                            onChange={this.onChange}
                        />
                        <label htmlFor="preferences.budget"
className="form-label">Procuro anúncios em torno de (R$):</label>
                    </div>
                    <div className="form-field">
                        <label className="form-label">Prefere casa ou
apartamento?</label>
                        <select
                            id="preferences.houseOrApartment"
                            name="houseOrApartment"
                            value={preferences.houseOrApartment}
                            onChange={this.onChange}
                            className="form-select"
                        >
                            <option value="casa">Casa</option>
                            <option value="apartamento">Apartamento</option>
                        </select>
                    </div>
                    <div className="form-field">
                        <label className="form-label">Procuro anúncios de aluguéis
destinados à:</label>
                        <select
                            id="preferences.genderPreference"
                            name="genderPreference"
                            value={preferences.genderPreference}
```

```
                        onChange={this.onChange}
                        className="form-select"
                    >
                        <option value="masculino">Masculino</option>
                        <option value="feminino">Feminino</option>
                        <option value="tanto faz">Tanto Faz</option>
                        <option value="ambos">Ambos</option>
                    </select>
                </div>
                <div className="form-field">
                    <label className="form-label">Aluguel
compartilhado?</label>
                    <select
                        id="preferences.roommates"
                        name="roommates"
                        value={preferences.roommates}
                        onChange={this.onChange}
                        className="form-select"
                    >
                        <option value="sozinho">Sozinho</option>
                        <option value="compartilhado">Compartilhado</option>
                    </select>
                </div>
                <div className="form-field">
                    <label className="form-label">Duração do aluguel</label>
                    <select
                        id="preferences.leaseLength"
                        name="leaseLength"
                        value={preferences.leaseLength}
                        onChange={this.onChange}
                        className="form-select"
                    >
                        <option value="aluguel anual">Aluguel anual</option>
                        <option value="aluguel temporada">Aluguel
temporada</option>
                    </select>
                </div>
                <div className="form-field">
                    <label className="form-label">Nível de barulho</label>
                    <select
                        id="preferences.noiseLevel"
                        name="noiseLevel"
                        value={preferences.noiseLevel}
                        onChange={this.onChange}
                        className="form-select"
                    >
                        <option value="tranquilo">Silencioso</option>
                        <option value="barulhento">Social</option>
                    </select>
                </div>
```

```
                        <div className="form-field">
                          <p className="checkbox-label">
                            <label>
                              <input
                                type="checkbox"
                                id="preferences.acceptsPets"
                                name="acceptsPets"
                                checked={preferences.acceptsPets}
                                onChange={this.onChange}
                              />
                              <span className="checkbox-text">Aceita Pets?</span>
                            </label>
                          </p>
                        </div>
                        <div className="form-field">
                          <p className="checkbox-label">
                            <label>
                              <input
                                type="checkbox"
                                id="preferences.wheelchairAccessible"
                                name="wheelchairAccessible"
                                checked={preferences.wheelchairAccessible}
                                onChange={this.onChange}
                              />
                              <span className="checkbox-text">Acessível à
cadeirantes?</span>
                            </label>
                          </p>
                        </div>
                        <div className="form-field">
                          <p className="checkbox-label">
                            <label>
                              <input
                                type="checkbox"
                                id="preferences.acceptSmoker"
                                name="acceptSmoker"
                                checked={preferences.acceptSmoker}
                                onChange={this.onChange}
                              />
                              <span className="checkbox-text">Fumante?</span>
                            </label>
                          </p>
                        </div>
                        <div className="form-field">
                          <p className="checkbox-label">
                            <label>
                              <input
                                type="checkbox"
                                id="preferences.hasFurniture"
                                name="hasFurniture"
```

```
                                    checked={preferences.hasFurniture}
                                    onChange={this.onChange}
                                />
                                <span className="checkbox-text">Local
mobiliado?</span>
                            </label>
                        </p>
                    </div>
                    <div className="form-button-container">
                        <button
                            className="btn-large-register"
                            type="submit"
                        >
                            Salvar
                        </button>
                    </div>
                </form>
            </div>
        </div>
    </div>
  );
 }
}

Register.propTypes = {
  registerUser: PropTypes.func.isRequired,
  auth: PropTypes.object.isRequired,
  errors: PropTypes.object.isRequired
};

const mapStateToProps = state => ({
  auth: state.auth,
  errors: state.errors
});

export default connect(
  mapStateToProps,
  { registerUser }
)(withRouter(Register));
```

CAMINHO: client/src/views/User/Register.css
ARQUIVO: Register.css

```
.register-container {
    padding-bottom: 50px;
}

.image-container {
    background-image:
url('https://media.gettyimages.com/id/1321503267/pt/foto/a-college-dorm-with-classic
-furniture-and-an-old-tv-in-the-morning.jpg?s=612x612&w=0&k=20&c=6ijRY0Kds8M
bNTIUFSCv-GCdysbN5s4vGAUWfhl71Lk=');
    background-size: cover;
    background-repeat: repeat repeat;
    border-radius: 0px;
    padding-bottom: 10px;
    margin-bottom: -54px;
}

.btn-back {
    margin-bottom: 20px;
}

.form-container {
    padding-left: 149.25px;
}

.form-container-login {
    margin-left: 365px;
}

.form-title-register {
    font-size: 24px;
    font-weight: bold;
    margin-bottom: 20px;
    margin-left: -256px;
    color: #000000;
    text-shadow: -1px -1px 0 #ffffff, 1px -1px 0 #ffffff, -1px 1px 0 #ffffff, 1px 1px 0 #ffffff;
}

.login-link {
    color: #333;
}

.form-field {
    margin-bottom: 22px;
    margin-left: 90px;
    margin-right: 20px;
}
```

```
.form-field-input {
    margin-bottom: 22px;
    margin-left: 90px;
    margin-right: 438px;
}

.form-field i.material-icons.prefix {
    position: absolute;
    top: 50%;
    transform: translateY(-50%);
    left: 15px;
    font-size: 24px;
}

.form-field label {
    font-size: 16px;
    color: #000000;
    text-shadow: -1px -1px 0 #ffffff, 1px -1px 0 #ffffff, -1px 1px 0 #ffffff, 1px 1px 0 #ffffff;
}

.form-field select {
    display: block;
    width: calc(100% - 40px);
    padding: 10px 10px 10px 40px;
    border: 1px solid #ccc;
    border-radius: 4px;
    font-size: 16px;
    color: #333;
}

.form-field label input[type="checkbox"] {
    position: absolute;
    opacity: 0;
}

.btn-flat-back {
    color: #2196F3;
    font-weight: bold;
    width: 100%;
    text-shadow: -1px -1px 0 #ffffff, 1px -1px 0 #ffffff, -1px 1px 0 #ffffff, 1px 1px 0 #ffffff;
}

.btn-large-register {
    height: auto;
    display: flex;
    align-items: center;
    border: none;
    border-radius: 4px;
    padding: 13px 44px;
    text-decoration: none;
```

```
    white-space: normal;
    margin-left: 500px;
    cursor: pointer;
    background-color: #007bff;
    color: white;
    font-size: 25px;
    font-weight: bold;
}

.form-label {
    font-size: 20px;
    color: #000000;
    font-weight: bolder;
    text-shadow: -1px -1px 0 #ffffff, 1px -1px 0 #ffffff, -1px 1px 0 #ffffff, 1px 1px 0 #ffffff;
}

@media (max-width: 600px) {
    .image-container {
        background-size: contain;
    }
    .form-container {
        padding-left: 20px;
    }
    .form-field, .form-field-input {
        margin-left: 20px;
        margin-right: 20px;
    }
    .form-field i.material-icons.prefix {
        left: 5px;
        font-size: 20px;
    }
    .form-label {
        font-size: 18px;
    }
    .form-field select {
        width: calc(100% - 10px);
        padding: 10px 10px 10px 20px;
        font-size: 14px;
    }
    .form-field label span {
        padding-left: 10px;
        font-size: 14px;
    }
    .btn-large-register {
        margin-left: 40px;
    }
}
@media (max-width: 1200px) {
    .image-container {
        background-size: contain;
```

```
    }
    .form-title-register {
        font-size: 20px;
        margin-left: -59px;
        margin-bottom: -130px;
    }
}

@media (max-width: 768px) {
    .form-field, .form-field-input {
        margin-left: 10px;
        margin-right: 10px;
    }
    .form-field i.material-icons.prefix {
        left: 5px;
        font-size: 18px;
    }
    .form-label {
        font-size: 16px;
    }
    .form-field select {
        width: calc(100% - 10px);
        padding: 10px 10px 10px 20px;
        font-size: 14px;
    }
    .form-field label span {
        padding-left: 10px;
        font-size: 14px;
    }
}

@media (max-width: 992px) {
    .form-field, .form-field-input {
        margin-left: 20px;
        margin-right: 20px;
    }
    .image-container {
        background-size: contain;
    }
}
```

CAMINHO: client/src/views/User/Login.js
ARQUIVO: Login.js

```
import React, { Component } from "react";
import { Link } from "react-router-dom";
import PropTypes from "prop-types";
import { connect } from "react-redux";
```

```
import { loginUser } from "../../actions/authActions.js";
import "./Login.css"; // Importa o arquivo CSS personalizado

class Login extends Component {
    constructor() {
        super();
        this.state = {
            email: "",
            password: "",
            errors: {},
        };
    }

    componentDidMount() {
        if (this.props.auth.isAuthenticated) {
            this.props.history.push("/dashboard");
        }
    }

    componentWillReceiveProps(nextProps) {
        if (nextProps.auth.isAuthenticated) {
            this.props.history.push("/dashboard");
        }
        if (nextProps.errors) {
            this.setState({
                errors: nextProps.errors,
            });
        }
    }

    onChange = (e) => {
        this.setState({ [e.target.id]: e.target.value });
    };

    onSubmit = (e) => {
        e.preventDefault();
        const userData = {
            email: this.state.email,
            password: this.state.password,
        };
        this.props.loginUser(userData);
    };

    render() {
        const { errors } = this.state;
        return (
            <div className="login-container">
                <div className="login-content">
                    <Link to="/" className="back-link">
                        Voltar ao início
```

```
                </Link>
                <div className="heading">
                  <h4>Faça o login</h4>
                  <p>
                    <b>Ainda não é cadastrado?</b>{" "}
                    <Link to="/register">Cadastre-se</Link>
                  </p>
                </div>
                <form noValidate onSubmit={this.onSubmit}>
                  <div className="input-field">
                    <input
                      onChange={this.onChange}
                      value={this.state.email}
                      error={errors.email || errors.emailnotfound}
                      id="email"
                      type="email"
                      placeholder="Email"
                    />
                    <span className="error-text">
                      {errors.email}
                      {errors.emailnotfound}
                    </span>
                  </div>
                  <div className="input-field">
                    <input
                      onChange={this.onChange}
                      value={this.state.password}
                      error={errors.password || errors.passwordincorrect}
                      id="password"
                      type="password"
                      placeholder="Password"
                    />
                    <span className="error-text">
                      {errors.password}
                      {errors.passwordincorrect}
                    </span>
                  </div>
                  <div className="button-container">
                    <button type="submit" className="login-button">
                      Login
                    </button>
                  </div>
                </form>
              </div>
            </div>
        );
    }
}

Login.propTypes = {
```

```
    loginUser: PropTypes.func.isRequired,
    auth: PropTypes.object.isRequired,
    errors: PropTypes.object.isRequired,
};

const mapStateToProps = (state) => ({
    auth: state.auth,
    errors: state.errors,
});

export default connect(mapStateToProps, { loginUser })(Login);
```

CAMINHO: client/src/views/User/Login.js
ARQUIVO: Login.js

```
.login-container {
    background-image:
url('https://media.gettyimages.com/id/83375946/pt/foto/college-dorm-hallway.jpg?s=6
12x612&w=0&k=20&c=KnkPhSTzkyEHq2YBUAPpV0cLeVCih4324ZrvIrfNBUI=');
    background-size: cover;
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-position: center;
    height: 100vh;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    text-align: center;
    border-radius: 5px;
    padding-top: 50px;
    padding-bottom: 50px;
}

.login-content {
    width: 100%;
    max-width: 400px;
    background: #fff;
    border-radius: 5px;
    padding: 20px;
    box-shadow: 0 0 10px rgb(0, 0, 0);
}

.back-link {
    text-decoration: none;
    font-weight: bold;
    color: #2196F3;
    display: block;
```

```
    margin-bottom: 20px;
}

.back-link:hover {
    text-decoration: underline;
}

.heading {
    text-align: center;
    margin-bottom: 20px;
}

.heading h4 {
    font-size: 24px;
    color: #000000;
    text-shadow: -1px -1px 0 #ffffff, 1px -1px 0 #ffffff, -1px 1px 0 #ffffff, 1px 1px 0 #ffffff;
}

.heading a {
    text-decoration: none;
    font-weight: bold;
    color: #2196F3;
}

.heading a:hover {
    text-decoration: underline;
}

.input-field {
    margin-bottom: 20px;
    position: relative;
    color: #000000;
}

.error-text {
    color: red;
    font-size: 14px;
    position: absolute;
    bottom: -20px;
    left: 0;
}

.login-button {
    background-color: #2196F3;
    color: #fff;
    font-weight: bold;
    width: 100%;
    padding: 10px;
    border: none;
    border-radius: 5px;
```

```
    cursor: pointer;
    transition: background-color 0.3s ease-in-out;
}

.login-button:hover {
    background-color: #1976D2;
}

.input-field input::placeholder {
    color: #000000;
}
```

CAMINHO: client/src/views/User/EditPreferences.js
ARQUIVO: EditPreferences.js

```
import React, { Component } from "react";
import { withRouter } from "react-router-dom";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import { updateUserPreferences } from "../../actions/authActions.js";
import "./EditPreferences.css";
import axios from "axios";

class EditPreferences extends Component {
    constructor(props) {
        super(props);
        this.state = {
            preferences: {
                houseOrApartment: "",
                genderPreference: "",
                acceptsPets: false,
                location: "",
                roommates: "",
                leaseLength: "",
                budget: "",
                wheelchairAccessible: false,
                noiseLevel: "",
                acceptSmoker: false,
                hasFurniture: false
            },
        };
    }

    componentDidMount() {
        // Busca preferências do usuário no banco de dados
        axios
            .get("/api/users/preferences")
            .then((res) => {
```

```
            const userPreferences = res.data;
            this.setState({ preferences: userPreferences });
        })
        .catch((err) => {
            console.error("Erro ao listar as preferências do usuário:", err);
        });
}

onChange = (e) => {
    const { name, value, type, checked } = e.target;
    this.setState((prevState) => ({
        preferences: {
            ...prevState.preferences,
            [name]: type === "checkbox" ? checked : value,
        },
    }));
};

onSubmit = (e) => {
    e.preventDefault();

    // Cria um objeto com as preferências atualizadas
    const updatedPreferences = {
        ...this.state.preferences,
    };

    // Despacha uma ação para atualizar as preferências do usuário
    this.props.updateUserPreferences(updatedPreferences, this.props.history);
};

render() {
    const { preferences } = this.state;
    return (
        <div className="edit-preferences-container">
            <div className="edit-preferences-form">
                <h4 className="form-title">
                    <b>Editar Preferências</b>
                </h4>
                <form noValidate onSubmit={this.onSubmit}>
                    <div className="form-field">
                        <label>Bairro de preferência</label>
                        <input
                            name="location"
                            type="text"
                            value={preferences.location}
                            onChange={this.onChange}
                        />
                    </div>
                    <div className="form-field">
                        <label>Valor de aluguel (R$):</label>
```

```
    <input
      name="budget"
      type="number"
      value={preferences.budget}
      onChange={this.onChange}
    />
  </div>
  <div className="form-field">
    <label>Prefere casa ou apartamento?</label>
    <select
      name="houseOrApartment"
      value={preferences.houseOrApartment}
      onChange={this.onChange}
    >
      <option value="casa">Casa</option>
      <option value="apartamento">Apartamento</option>
    </select>
  </div>
  <div className="form-field">
    <label>Procuro anúncios de aluguéis destinados à:</label>
    <select
      name="genderPreference"
      value={preferences.genderPreference}
      onChange={this.onChange}
    >
      <option value="masculino">Masculino</option>
      <option value="feminino">Feminino</option>
      <option value="tanto faz">Tanto Faz</option>
      <option value="ambos">Ambos</option>
    </select>
  </div>
  <div className="form-field">
    <label>Aluguel compartilhado?</label>
    <select
      name="roommates"
      value={preferences.roommates}
      onChange={this.onChange}
    >
      <option value="sozinho">Sozinho</option>
      <option value="compartilhado">Compartilhado</option>
    </select>
  </div>
  <div className="form-field">
    <label>Duração do aluguel</label>
    <select
      name="leaseLength"
      value={preferences.leaseLength}
      onChange={this.onChange}
    >
      <option value="aluguel anual">Aluguel anual</option>
```

```
                        <option value="aluguel temporada">Aluguel temporada</option>
                    </select>
                </div>
                <div className="form-field">
                    <label>Nível de barulho</label>
                    <select
                        name="noiseLevel"
                        value={preferences.noiseLevel}
                        onChange={this.onChange}
                    >
                        <option value="tranquilo">Silencioso</option>
                        <option value="barulhento">Social</option>
                    </select>
                </div>
                <div className="input-field col s12 form-field">
                    <p className="checkbox-label">
                        <label>
                            <input
                                type="checkbox"
                                id="preferences.acceptsPets"
                                name="acceptsPets"
                                checked={preferences.acceptsPets}
                                onChange={this.onChange}
                            />
                            <span className="checkbox-text">Aceita Pets?</span>
                        </label>
                    </p>
                </div>
                <div className="input-field col s12 form-field">
                    <p className="checkbox-label">
                        <label>
                            <input
                                type="checkbox"
                                id="preferences.wheelchairAccessible"
                                name="wheelchairAccessible"
                                checked={preferences.wheelchairAccessible}
                                onChange={this.onChange}
                            />
                                        <span className="checkbox-text">Acessível à
cadeirantes?</span>
                        </label>
                    </p>
                </div>
                <div className="input-field col s12 form-field">
                    <p className="checkbox-label">
                        <label>
                            <input
                                type="checkbox"
                                id="preferences.acceptSmoker"
                                name="acceptSmoker"
```

```
                                checked={preferences.acceptSmoker}
                                onChange={this.onChange}
                            />
                            <span className="checkbox-text">Fumante?</span>
                        </label>
                    </p>
                </div>
                <div className="form-field">
                    <p className="checkbox-label">
                        <label>
                            <input
                                type="checkbox"
                                id="preferences.hasFurniture"
                                name="hasFurniture"
                                checked={preferences.hasFurniture}
                                onChange={this.onChange}
                            />
                            <span className="checkbox-text">Local mobiliado?</span>
                        </label>
                    </p>
                </div>
                <div className="form-container-preferences">
                    <button className="btn-update" type="submit">
                        Atualizar
                    </button>
                </div>
            </form>
        </div>
    </div>
  );
 }
}

EditPreferences.propTypes = {
 updateUserPreferences: PropTypes.func.isRequired,
 auth: PropTypes.object.isRequired,
};

const mapStateToProps = (state) => ({
 auth: state.auth,
});

export default connect(mapStateToProps, { updateUserPreferences })(
 withRouter(EditPreferences)
);
```

CAMINHO: client/src/views/User/EditPreferences.css
ARQUIVO: EditPreferences.css

```
.edit-preferences-container {
    padding-top: 50px;
    padding-bottom: 50px;
    background-image:
url('https://media.gettyimages.com/id/1317794799/pt/foto/interior-view-of-a-dormitory
-at-ludgrove-school-an-independent-preparatory-boarding-school-in.jpg?s=612x612
&w=0&k=20&c=smdb9Mm4Cob81lalCjFOKF_frGQucRGLqsJntt7WY2w=');
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
}

.edit-preferences-form {
    padding-left: 11.250px;
    margin-left: 180px;
}

.form-title {
    font-size: 24px;
    font-weight: bold;
    margin-bottom: 20px;
    margin-left: -120px;
    color: #000000;
    text-shadow: -1px -1px 0 #ffffff, 1px -1px 0 #ffffff, -1px 1px 0 #ffffff, 1px 1px 0 #ffffff;
/* Add text shadow for black outline */
}

.form-field {
    margin-bottom: 20px;
    margin-left: 60px;
}

.form-field label {
    font-size: 16px;
    color: #000000;
    text-shadow: -1px -1px 0 #ffffff, 1px -1px 0 #ffffff, -1px 1px 0 #ffffff, 1px 1px 0 #ffffff;
/* Add text shadow for black outline */
}

.form-field input[type="text"],
.form-field input[type="number"],
.form-field select {
    display: block;
    width: 70%;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 4px;
```

```
    font-size: 16px;
    font-weight: bold;
    color: #fff; /* Set text color to white */
    text-shadow: -1px -1px 0 #000, 1px -1px 0 #000, -1px 1px 0 #000, 1px 1px 0
#000; /* Add text shadow for black outline */
}

.form-field .checkbox-label input[type="checkbox"] {
    position: absolute;
    opacity: 0;
}

.form-field label span {
    padding-left: 30px;
    font-size: 16px;
    color: #333;
}

.checkbox-label {
    display: flex;
    align-items: center;
    font-size: 16px;
    color: #000000; /* Set text color to white */
    text-shadow: -1px -1px 0 #ffffff, 1px -1px 0 #ffffff, -1px 1px 0 #ffffff, 1px 1px 0 #ffffff;
/* Add text shadow for black outline */
}

.checkbox-text {
    padding-left: 10px;
    font-weight: bold;
    color: #000000;
}

.form-field select {
    background-color: #852b2b;
}

.btn-update {
    background-color: #007bff;
    font-size: 18px;
    border: none;
    border-radius: 4px;
    padding: 10px 20px;
    cursor: pointer;
    margin-left: 381px;
    color: #000000;
    text-shadow: -1px -1px 0 #ffffff, 1px -1px 0 #ffffff, -1px 1px 0 #ffffff, 1px 1px 0 #ffffff;
}
```

```
@media screen and (max-width: 576px) {
    .edit-preferences-container {
        padding-top: 20px;
        padding-bottom: 18px;
        margin-left: -153px;
        margin-right: -12px;
    }

    .form-title {
        font-size: 20px;
        margin-left: -80px;
    }

    .form-field {
        margin-left: 30px;
    }

    .form-field label {
        font-size: 14px;
    }

    .form-field input[type="text"],
    .form-field input[type="number"],
    .form-field select {
        padding: 8px;
        font-size: 14px;
    }

    .form-field label span {
        font-size: 14px;
    }

    .btn-update {
        font-size: 16px;
        padding: 8px 16px;
        margin-left: 71px;
    }
}

@media screen and (min-width: 577px) and (max-width: 768px) {
    .edit-preferences-container {
        padding-top: 30px;
        padding-bottom: 30px;
    }

    .form-title {
        font-size: 22px;
        margin-left: -100px;
    }
```

```
    .form-field {
        margin-left: 45px;
    }

    .form-field label {
        font-size: 16px;
    }

    .form-field input[type="text"],
    .form-field input[type="number"],
    .form-field select {
        padding: 10px;
        font-size: 16px;
    }

    .form-field label span {
        font-size: 16px;
    }

    .btn-update {
        font-size: 18px;
        padding: 10px 20px;
    }
}

@media screen and (min-width: 769px) and (max-width: 1024px) {
    .edit-preferences-container {
        padding-top: 40px;
        padding-bottom: 40px;
    }

    .form-title {
        font-size: 24px;
        margin-left: -120px;
    }

    .form-field {
        margin-left: 60px;
    }

    .form-field label {
        font-size: 18px;
    }

    .form-field input[type="text"],
    .form-field input[type="number"],
    .form-field select {
        padding: 12px;
        font-size: 18px;
    }
```

```
    .form-field label span {
        font-size: 18px;
    }

    .btn-update {
        font-size: 20px;
        padding: 12px 24px;
    }
}

@media screen and (min-width: 1025px) and (max-width: 1440px) {
    .edit-preferences-container {
        padding-top: 50px;
        padding-bottom: 50px;
    }

    .form-title {
        font-size: 26px;
        margin-left: -140px;
    }

    .form-field {
        margin-left: 75px;
    }

    .form-field label {
        font-size: 20px;
    }

    .form-field input[type="text"],
    .form-field input[type="number"],
    .form-field select {
        padding: 14px;
        font-size: 20px;
    }

    .form-field label span {
        font-size: 20px;
    }

    .btn-update {
        font-size: 22px;
        padding: 14px 28px;
    }
}

@media screen and (min-width: 1441px) and (max-width: 1680px) {
    .edit-preferences-container {
        padding-top: 60px;
```

```
        padding-bottom: 60px;
    }

    .form-title {
        font-size: 28px;
        margin-left: -160px;
    }

    .form-field {
        margin-left: 90px;
    }

    .form-field label {
        font-size: 22px;
    }

    .form-field input[type="text"],
    .form-field input[type="number"],
    .form-field select {
        padding: 6px;
        font-size: 22px;
    }

    .form-field label span {
        font-size: 22px;
    }

    .btn-update {
        font-size: 24px;
        padding: 16px 32px;
    }
}
```

CAMINHO: client/src/views/layout/Navbar.js
ARQUIVO: Navbar.js

```
import React, { Component } from "react";
import { Link, withRouter } from "react-router-dom";
import { connect } from "react-redux";
import { logoutUser } from "../../actions/authActions.js";
import "./Navbar.css";

class Navbar extends Component {
    // Função chamada ao clicar no botão de logout
    onLogoutClick = (e) => {
```

```
        e.preventDefault();
        this.props.logoutUser();
    };

    render() {
        // Extrai informações sobre autenticação e usuário do estado Redux
        const { isAuthenticated, user } = this.props.auth;

        return (
            <div className="navbar">
                <nav className="z-depth-0">
                    <div className="nav-wrapper">
                        {/* Link para a página inicial */}
                        <Link to="/" className="col s5 brand-logo center black-text">
                            <b className="home-link"> Início</b>
                        </Link>

                        {/* Botões de navegação condicionais com base na autenticação */}
                        <div className="nav-buttons">
                            {isAuthenticated && (
                                <Link to="/preferences" className="btn-edit-prefs">
                                    <p>Editar Preferências</p>
                                </Link>
                            )}
                            {isAuthenticated && (
                                                                <Link   to="/recommendation"
classNameName="btn-show-recommendations">
                                    <p>Recomendações</p>
                                </Link>
                            )}
                        </div>

                        {/* Se autenticado, exibe informações do usuário e botão de logout */}
                        {isAuthenticated ? (
                            <div className="user-info">
                                <span className="user-email">{user.email}</span>
                                <button
                                    onClick={this.onLogoutClick}
                                    className="btn btn-logout"
                                >
                                    <b>Logout</b>
                                </button>
                            </div>
                        ) : null}
                    </div>
                </nav>
            </div>
        );
    }
}
```

```
// Mapeia o estado do Redux para as propriedades do componente
const mapStateToProps = (state) => ({
    auth: state.auth,
});

// Conecta o componente ao Redux e ao Router
export default withRouter(connect(mapStateToProps, { logoutUser })(Navbar));
```

CAMINHO: client/src/views/layout/Navbar.css
ARQUIVO: Navbar.css

```
.navbar {
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.home-link{
    margin-left: -32px;
}
.nav-wrapper {
    background-color: #68917d;
    display: flex;
    justify-content: space-between;
    align-items: center;
    height: 48px;
}

.btn-edit-prefs,
.btn-show-recommendations {
    height: auto;
    display: flex;
    align-items: center;
    background-color: #041f3b;
    color: #fff;
    border: none;
    border-radius: 4px;
    padding: 4px 6px;
    text-decoration: none;
    white-space: normal;
    font-size: 0.9rem;
    margin-left: 0px;
    margin-top: 10px;
}

.btn-edit-prefs p,
.btn-show-recommendations p {
    margin: 0;
    padding: 0;
```

```
    line-height: 1;
    white-space: normal;
    font-size: 0.8rem;
}

.user-info {
    display: flex;
    flex-direction: column;
    align-items: center;
    color: #000;
    font-weight: bold;
    white-space: nowrap;
    overflow: hidden;
}

.user-email {
    margin-bottom: -38px;
    font-size: 0.8rem;
}

.btn-logout {
    background-color: #9d0101;
    padding: 6px 8px;
    line-height: 1;
    white-space: nowrap;
    font-size: 0.8rem;
    margin-bottom: 20px;
}

@media (max-width: 768px) {
    .nav-wrapper {
        height: auto;
    }

    .material-icons {
        font-size: 0.8rem;
        margin-right: 4px;
    }

    .col.s5.brand-logo.center.black-text,
    .col.s5.brand-logo.black-text {
        font-size: 1rem;
        max-height: none;
        overflow: visible;
        white-space: normal;
    }

    .btn-edit-prefs,
    .btn-show-recommendations,
    .btn-logout {
```

```
        height: auto;
        padding: 6px 8px;
        margin-left: 0px;

    }

    .btn-edit-prefs p,
    .btn-show-recommendations p,
    .btn-logout p {
        font-size: 0.8rem;
    }
}

@media (max-width: 320px) and (max-height: 668px) {
    .nav-wrapper {
        height: 44px;
    }

    .material-icons {
        font-size: 0.8rem;
        margin-right: 4px;
    }

    .col.s5.brand-logo.center.black-text,
    .col.s5.brand-logo.black-text {
        font-size: 2rem;
        max-height: 36px;
    }

    .btn-edit-prefs,
    .btn-show-recommendations,
    .btn-logout {
        height: auto;
        padding: 4px 8px;
    }

    .btn-edit-prefs p,
    .btn-show-recommendations p,
    .btn-logout p {
        font-size: 0.6rem;
    }

    .user-email {
        font-size: 0.6rem;
    }
}

@media (max-width: 375px) {
    .btn-show-recommendations {
        font-size: 0.6rem;
```

```
    }
}

@media (min-width: 376px) and (max-width: 414px) {
    .btn-show-recommendations {
        font-size: 0.7rem;
        margin-right: auto;
    }
}

@media (min-width: 768px) and (max-width: 1024px) {
    .btn-show-recommendations {
        font-size: 0.9rem;
    }
}

@media (min-width: 1025px) and (max-width: 1440px) {
    .btn-show-recommendations {
        font-size: 1rem;
    }
}

@media (min-width: 1441px) {
    .btn-show-recommendations {
        font-size: 1.1rem;
    }
}
```

CAMINHO: client/src/views/layout/MainPage.js
ARQUIVO: MainPage.js

```
import React, { Component } from "react";
import { Link } from "react-router-dom";
import { connect } from "react-redux";
import "./MainPage.css"; // Importa os estilos CSS para a MainPage

class MainPage extends Component {
    render() {
        const { isAuthenticated } = this.props.auth;

        return (
            <div className="container-valign">
                <div className="row">
                    <div className="col s12 center-align">
                        <h4 className="title">
                            <b>Página Principal</b>
                        </h4>
                        <p className="description">
```

```
                    Portal de anúncios de aluguéis em Florianópolis indexados a partir
de técnicas de WebScrapping
                </p>
                <p className="description">INE - UFSC - 2023</p>
                <br />
                {isAuthenticated ? (
                    <div className="main-buttons">
                        <Link to="/products" className="btn btn-produtos">
                            <b>Anúncios</b>
                        </Link>
                        <Link to="/me" className="btn btn-profile">
                            <b>Perfil</b>
                        </Link>
                    </div>
                ) : (
                    <div className="main-buttons">
                        <Link to="/register" className="btn btn-cadastro">
                            <b>Cadastro</b>
                        </Link>
                        <Link to="/login" className="btn btn-login">
                            <b>Log In</b>
                        </Link>
                    </div>
                )}
            </div>
        </div>
    </div>
    );
  }
}

const mapStateToProps = (state) => ({
  auth: state.auth,
});

export default connect(mapStateToProps)(MainPage);
```

CAMINHO: client/src/views/layout/MainPage.css
ARQUIVO: MainPage.css

```
.container-valign {
  height: 95vh;
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
  background-image:
url('https://media.gettyimages.com/id/1335298641/pt/foto/university-dorm-room.jpg?s
=612x612&w=0&k=20&c=pU32Yo1uwPq4Tu0W9K1B9d94gp0Q8OcOIamG0cCRJJg
```

```
=');
    background-size: cover;
    background-position: center;
}

.row .col.s6 {
    width: 50%;
    margin-left: 190px;
}

.title {
    font-size: 24px;
    text-align: center;
    margin-bottom: 20px;
}

.description {
    text-align: center;
    font-size: 16px;
    color: #000000;
    font-weight: bold;
}

.button {
    display: inline-block;
    border: none;
    border-radius: 4px;
    padding: 0px 10px;
    text-decoration: none;
    margin: 10px;
    cursor: pointer;
    background-color: #007bff;
    color: white;
}

.btn-cadastro {
    background-color: #367306;
    margin-right: 10px;
}

.btn-login {
    background-color: #041f3b;
}

.btn-produtos {
    background-color: #afa74c;
    margin-right: 10px;
}

.btn-profile {
```

```
    background-color: #f39c12;
}

.main-buttons {
    margin-left: 10px;
}


@media screen and (max-width: 576px) {
    .container-valign {
        height: 96vh;
    }

    .button {
        padding: 8px 16px;
        font-size: 0.9rem;
    }

    .row .col.s6 {
        width: 50%;
        margin-left: 86px;
    }
}

@media screen and (min-width: 577px) and (max-width: 992px) {
    .container-valign {
        height: 90vh;
    }

    .button {
        padding: 10px 18px;
        font-size: 1rem;
    }

    .row .col.s6 {
        width: 50%;
        margin-left: 86px;
    }
}

@media screen and (min-width: 993px) {
    .container-valign {
        height: 100vh;
    }

    .button {
        padding: 12px 20px;
        font-size: 1.1rem;
    }
```

```
    .row .col.s6 {
        width: 50%;
        margin-left: 86px;
    }
}

@media screen and (min-width: 768px) and (max-width: 1024px) {
    .container-valign {
        height: 85vh;
    }

    .row .col.s6 {
        width: 50%;
        margin-left: 86px;
    }
}
```

CAMINHO: client/src/views/layout/Landing.js
ARQUIVO: Landing.js

```
import React, { Component } from "react";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import { logoutUser } from "../../actions/authActions.js";
import { Link } from "react-router-dom";
import "./Landing.css"; // Importa o arquivo CSS atualizado

class Landing extends Component {
    render() {
        const { user } = this.props.auth;

        return (
            <div className="landing-container"> {/* Aplicar a imagem de fundo */}
                <div className="welcome-text">
                    <h4>
                        <b>Seja bem-vindo(a),</b> {user.name.split(" ")[0]}
                    </h4>
                    <p className="welcome-text">
                        <b>Você está logado(a) no VagouAqui :D</b>
                    </p>
                    <div className="col s6">
                        <Link
                            to="/products"
                            className="btn produtos-btn"
                        >
                            Anúncios
                        </Link>
                    </div>
```

```
            </div>
          </div>
        );
    }
}

Landing.propTypes = {
    logoutUser: PropTypes.func.isRequired,
    auth: PropTypes.object.isRequired,
};

const mapStateToProps = (state) => ({
    auth: state.auth,
});

export default connect(mapStateToProps, { logoutUser })(Landing);
```

CAMINHO: client/src/views/layout/Landing.css
ARQUIVO: Landing.css

```
.container {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
}


h4 {
    font-size: 24px;
    text-align: center;
}

.flow-text {
    font-size: 18px;
    text-align: center;
    color: #000000;
    margin-top: 10px;
}


.btn {
    width: 140px;
    border-radius: 3px;
    letter-spacing: 1.5px;
    text-align: center;
    margin-top: 20px;
```

```
    cursor: pointer;
}


.produtos-btn {
    background-color: #041f3b;
}


.logout-btn {
    background-color: #FF5733;
    color: white;
}

.landing-container {
    background-image:
url('https://media.gettyimages.com/id/1335298641/pt/foto/university-dorm-room.jpg?s
=612x612&w=0&k=20&c=pU32Yo1uwPq4Tu0W9K1B9d94gp0Q8OcOIamG0cCRJJg
=');
    background-size: cover;
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-position: center;
    opacity: 0.9; /* Adjust the opacity as needed */
    height: 100vh;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    text-align: center;
}


@media (max-width: 768px) {

    h4 {
        font-size: 20px;
    }
    .flow-text {
        font-size: 16px;
    }
    .btn {
        width: 120px;
    }
}
```

CAMINHO: client/src/views/adGrid/RecommendedGrid.js
ARQUIVO: RecommendedGrid.js

```
import React, { useState, useEffect } from 'react';
import './RecommendedGrid.css';
import { Carousel } from 'react-responsive-carousel';
import 'react-responsive-carousel/lib/styles/carousel.min.css';
import Lightbox from 'react-image-lightbox';
import 'react-image-lightbox/style.css';
import axios from 'axios';

function RecommendedGrid() {
    const [ads, setAds] = useState([]);
    const [currentPage, setCurrentPage] = useState(1);
    const [adsPerPage] = useState(15);
    const [lightboxImages, setLightboxImages] = useState([]);
    const [lightboxImageIndex, setLightboxImageIndex] = useState(-1);

    useEffect(() => {
        axios
            .get('/api/recommendation/all')
            .then((res) => {
                const recommendedAds = res.data;
                console.log(recommendedAds);
                setAds(recommendedAds);
            })
            .catch((err) => {
                console.error('Erro ao listar anúncios recomendados:', err);
            });
    }, []); // Certificar-se de passar uma matriz de dependências vazia para executar
este efeito apenas uma vez

    const totalPages = Math.ceil(ads.length / adsPerPage);

    // Calcula os limites inferior e superior para os índices de página exibidos
    const pageRange = 10; // Número de índices de página a serem mostrados

    let lowerBound = Math.max(currentPage - Math.floor(pageRange / 2), 1);
    let upperBound = Math.min(lowerBound + pageRange - 1, totalPages);

    if (upperBound - lowerBound + 1 < pageRange) {
        lowerBound = Math.max(1, upperBound - pageRange + 1);
    }

    // Cria um array de índices de página para exibir
    const pageIndicesToDisplay = Array.from({ length: upperBound - lowerBound + 1
}, (_, i) => i + lowerBound);

    const paginate = pageNumber => setCurrentPage(pageNumber);
```

```
    const openLightbox = (images, imageIndex) => {
        setLightboxImages(images);
        setLightboxImageIndex(imageIndex);
    };

    const closeLightbox = () => {
        setLightboxImages([]);
        setLightboxImageIndex(-1);
    };

    const nextPage = () => {
        if (currentPage < totalPages) {
            paginate(currentPage + 1);
        }
    };

    const prevPage = () => {
        if (currentPage > 1) {
            paginate(currentPage - 1);
        }
    };

    const advanceTenPages = () => {
        const newPage = Math.min(currentPage + 10, totalPages);
        paginate(newPage);
    };

    const retreatTenPages = () => {
        const newPage = Math.max(currentPage - 10, 1);
        paginate(newPage);
    };

    return (
        <div>
            <div className="grid-container">
                <div className="grid">
                    {ads.slice((currentPage - 1) * adsPerPage, currentPage *
adsPerPage).map((ad, adIndex) => (
                        <div key={adIndex} className="grid-item">
                            <h2>{ad.ad.title}</h2>
                            <p className="contact-info">
                                <h2>Contato: {ad.ad.contactInfo}</h2>
                            </p>
                            <Carousel showArrows={true} infiniteLoop={true}>
                                {ad.ad.imageLinks.map((imageLink, imgIndex) => (
                                    <div key={imgIndex} onClick={() =>
openLightbox(ad.ad.imageLinks, imgIndex)}>
                                        <img src={imageLink} alt={`Ad ${imgIndex}`}
className="ad-image" />
                                    </div>
```

```
                        ))}
                    </Carousel>
                    <p className="ad-description">{ad.ad.description}</p>
                    <p className="ad-price">
                        <h3>{ad.ad.price}</h3>
                    </p>
                    <p className="ad-neighborhood">
                        <h3>{ad.ad.neighborhood}</h3>
                    </p>
                </div>
            ))}
        </div>
        <div className="pagination">
            <button onClick={prevPage} disabled={currentPage ===
1}>Anterior</button>
            <button onClick={retreatTenPages} disabled={currentPage - 10 < 1}>-10
Páginas</button>
            {pageIndicesToDisplay.map((pageIndex) => (
                <button
                    key={pageIndex}
                    onClick={() => paginate(pageIndex)}
                    className={currentPage === pageIndex ? 'active' : ''}
                >
                    {pageIndex}
                </button>
            ))}
            <button onClick={advanceTenPages} disabled={currentPage + 10 >
totalPages}>+10 Páginas</button>
            <button onClick={nextPage} disabled={currentPage ===
totalPages}>Próximo</button>
        </div>
    </div>
    {lightboxImages.length > 0 && lightboxImageIndex !== -1 && (
        <Lightbox
            mainSrc={lightboxImages[lightboxImageIndex]}
            nextSrc={lightboxImages[(lightboxImageIndex + 1) %
lightboxImages.length]}
            prevSrc={lightboxImages[(lightboxImageIndex + lightboxImages.length -
1) % lightboxImages.length]}
            onCloseRequest={closeLightbox}
            onMovePrevRequest={() =>
setLightboxImageIndex((lightboxImageIndex + lightboxImages.length - 1) %
lightboxImages.length)}
            onMoveNextRequest={() =>
setLightboxImageIndex((lightboxImageIndex + 1) % lightboxImages.length)}
        />
    )}
  </div>
 );
}
```

```
export default RecommendedGrid;
```

CAMINHO: client/src/views/adGrid/RecommendedGrid.css
ARQUIVO: RecommendedGrid.css

```
.grid-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin: 20px;
}

.grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
    grid-gap: 20px;
}

.grid-item {
    border: 1px solid #ddd;
    padding: 20px;
    text-align: center;
    background-color: #f5f5f5;
    max-height: 300px;
    overflow-y: auto;
    word-wrap: break-word;
}

.grid-item h2 {
    font-size: 1.2em;
    font-weight: bold;
    margin: 0;
}

.grid-item .contact-info {
    font-size: 1em;
    margin-top: 10px;
}

.carousel-container {
    max-width: 100%;
    height: auto;
    overflow: hidden;
}

.carousel .slider-wrapper {
    display: flex;
    align-items: center;
```

```
}

.ad-price {
    font-size: 1.2em;
    font-weight: bold;
    margin-top: 10px;
}

.ad-neighborhood {
    font-size: 0.9em;
    font-weight: bold;
    margin-top: 10px;
}

.pagination {
    display: flex;
    justify-content: center;
    margin-top: 20px;
}

.pagination button {
    margin: 0 5px;
    padding: 5px 10px;
    background-color: #f0f0f0;
    border: none;
    cursor: pointer;
}

.pagination button.active {
    background-color: #3498db;
    color: white;
}

@media screen and (max-width: 768px) {
    .grid {
        grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    }
    .grid-item {
        padding: 15px;
    }
    .carousel-container {
        height: auto;
    }
    .pagination button {
        padding: 4px 8px;
    }
}

@media screen and (max-width: 320px) and (max-height: 668px) {
    .grid {
```

```
        grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));
    }
    .grid-item {
        padding: 10px;
        max-height: none;
    }
    .grid-item h2 {
        font-size: 1em;
    }
    .carousel-container {
        height: auto;
    }
    .pagination button {
        padding: 4px 6px;
        font-size: 0.8em;
    }
}
```

CAMINHO: client/src/views/adGrid/AdGrid.js
ARQUIVO: AdGrid.js

```
import React, { useState, useEffect } from 'react';
import './AdGrid.css';
import { Carousel } from 'react-responsive-carousel';
import 'react-responsive-carousel/lib/styles/carousel.min.css';
import Lightbox from 'react-image-lightbox';
import 'react-image-lightbox/style.css';
import axios from 'axios';

function AdGrid() {
    const [ads, setAds] = useState([]);
    const [lastScrapingDate, setLastScrapingDate] = useState([]); // Novo estado para
a última data de scraping
    const [currentPage, setCurrentPage] = useState(1);
    const [adsPerPage] = useState(15);
    const [lightboxImages, setLightboxImages] = useState([]);
    const [lightboxImageIndex, setLightboxImageIndex] = useState(-1);

    useEffect(() => {
        async function fetchAds() {
            try {
                const response = await axios.get('/api/ads/all');
                const data = response.data;
                setAds(data);

                const lastScrapingDateResponse = await
axios.get('/api/ads/lastScrapingDate'); // Busca a última data de scraping
                setLastScrapingDate(lastScrapingDateResponse.data.lastScrapingDate);
```

```
// Define a última data de scraping
            } catch (error) {
                console.error('Erro ao listar anúncios:', error);
            }
        }
        fetchAds();
    }, []);

    const totalPages = Math.ceil(ads.length / adsPerPage);

    // Calcula os limites inferior e superior para os índices de página exibidos
    const pageRange = 10; // Número de índices de página a serem mostrados

    let lowerBound = Math.max(currentPage - Math.floor(pageRange / 2), 1);
    let upperBound = Math.min(lowerBound + pageRange - 1, totalPages);

    if (upperBound - lowerBound + 1 < pageRange) {
        lowerBound = Math.max(1, upperBound - pageRange + 1);
    }

    // Cria um array de índices de página para exibir
    const pageIndicesToDisplay = Array.from({ length: upperBound - lowerBound + 1
}, (_, i) => i + lowerBound);

    const paginate = pageNumber => setCurrentPage(pageNumber);

    const openLightbox = (images, imageIndex) => {
        setLightboxImages(images);
        setLightboxImageIndex(imageIndex);
    };

    const closeLightbox = () => {
        setLightboxImages([]);
        setLightboxImageIndex(-1);
    };

    const nextPage = () => {
        if (currentPage < totalPages) {
            paginate(currentPage + 1);
        }
    };

    const prevPage = () => {
        if (currentPage > 1) {
            paginate(currentPage - 1);
        }
    };

    const advanceTenPages = () => {
        const newPage = Math.min(currentPage + 10, totalPages);
```

```
    paginate(newPage);
  };

  const retreatTenPages = () => {
    const newPage = Math.max(currentPage - 10, 1);
    paginate(newPage);
  };

  return (
    <div>
      <div className="last-scraping-date">
        Anúncios atualizados pela última vez em: {lastScrapingDate}
      </div>
      <div className="grid-container">
        <div className="grid">
          {ads.slice((currentPage - 1) * adsPerPage, currentPage *
adsPerPage).map((ad, adIndex) => (
            <div key={adIndex} className="grid-item">
              <h2>{ad.title}</h2>
              <p className="contact-info">
                <h2>Contato: {ad.contactInfo}</h2>
              </p>
              <Carousel showArrows={true} infiniteLoop={true}>
                {ad.imageLinks.map((imageLink, imgIndex) => (
                  <div
                    key={imgIndex}
                    onClick={() => openLightbox(ad.imageLinks, imgIndex)}
                  >
                    <img src={imageLink} alt={`Ad ${imgIndex}`}
                    </div>
                ))}
              </Carousel>
              <p className="ad-description">{ad.description}</p>
              <p className="ad-price">
                <h3>{ad.price}</h3>
              </p>
              <p className="ad-neighborhood">
                <h3>{ad.neighborhood}</h3>
              </p>
            </div>
          ))}
        </div>
        <div className="pagination">
          <button onClick={prevPage} disabled={currentPage ===
1}>Anterior</button>
          <button onClick={retreatTenPages} disabled={currentPage - 10 < 1}>-10
Páginas</button>
          {pageIndicesToDisplay.map((pageIndex) => (
            <button
```

```
                        key={pageIndex}
                        onClick={() => paginate(pageIndex)}
                        className={currentPage === pageIndex ? 'active' : ''}
                    >
                        {pageIndex}
                    </button>
                ))}
                <button onClick={advanceTenPages} disabled={currentPage + 10 >
totalPages}>+10 Páginas</button>
                <button onClick={nextPage} disabled={currentPage ===
totalPages}>Próximo</button>
            </div>
        </div>
        {lightboxImages.length > 0 && lightboxImageIndex !== -1 && (
            <Lightbox
                mainSrc={lightboxImages[lightboxImageIndex]}
                nextSrc={lightboxImages[(lightboxImageIndex + 1) %
lightboxImages.length]}
                prevSrc={lightboxImages[(lightboxImageIndex + lightboxImages.length -
1) % lightboxImages.length]}
                onCloseRequest={closeLightbox}
                onMovePrevRequest={() =>
setLightboxImageIndex((lightboxImageIndex + lightboxImages.length - 1) %
lightboxImages.length)}
                onMoveNextRequest={() =>
setLightboxImageIndex((lightboxImageIndex + 1) % lightboxImages.length)}
            />
        )}
    </div>
  );
}
export default AdGrid;
```

CAMINHO: client/src/views/adGrid/AdGrid.css
ARQUIVO: AdGrid.css

```
.grid-container {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin: 20px;
}

.grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
  grid-gap: 20px;
}
```

```
.last-scraping-date {
    color: #367306;
    font-weight: bold;
    font-size: 18px;
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.5);
}

.grid-item {
    border: 1px solid #ddd;
    padding: 20px;
    text-align: center;
    background-color: #f5f5f5;
    max-height: 300px;
    overflow-y: auto;
    word-wrap: break-word;
}

.grid-item h2 {
    font-size: 1.2em;
    font-weight: bold;
    margin: 0;
}

.grid-item .contact-info {
    font-size: 1em;
    margin-top: 10px;
}

.carousel-container {
    max-width: 100%;
    height: auto;
    overflow: hidden;
}

.carousel .slider-wrapper {
    display: flex;
    align-items: center;
}

.ad-price {
    font-size: 1.2em;
    font-weight: bold;
    margin-top: 10px;
}

.ad-neighborhood {
    font-size: 0.9em;
    font-weight: bold;
    margin-top: 10px;
}
```

```
.pagination {
    display: flex;
    justify-content: center;
    margin-top: 20px;
}

.pagination button {
    margin: 0 5px;
    padding: 5px 10px;
    background-color: #f0f0f0;
    border: none;
    cursor: pointer;
}

.pagination button.active {
    background-color: #3498db;
    color: white;
}

@media screen and (max-width: 768px) {
    .grid {
        grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    }
    .grid-item {
        padding: 15px;
    }
    .carousel-container {
        height: auto;
    }
    .pagination button {
        padding: 4px 8px;
    }
}

@media screen and (max-width: 320px) and (max-height: 668px) {
    .grid {
        grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));
    }
    .grid-item {
        padding: 10px;
        max-height: none;
    }
    .grid-item h2 {
        font-size: 1em;
    }
    .carousel-container {
        height: auto;
    }
    .pagination button {
```

```
        padding: 4px 6px;
        font-size: 0.8em;
    }
}
```

CAMINHO: client/src/utils/PrivateRoute.js
ARQUIVO: PrivateRoute.js

```
import React from "react";
import { Route, Redirect } from "react-router-dom";
import { connect } from "react-redux";
import PropTypes from "prop-types";

// Componente PrivateRoute - uma rota protegida que redireciona para a página de
login se o usuário não estiver autenticado
const PrivateRoute = ({ component: Component, auth, ...rest }) => (
    <Route
        {...rest}
        render={props =>
            auth.isAuthenticated === true ? (
                // Se autenticado, renderiza o componente passando as props
                <Component {...props} />
            ) : (
                // Se não autenticado, redireciona para a página de login
                <Redirect to="/login" />
            )
        }
    />
);
// Define os tipos esperados das propriedades
PrivateRoute.propTypes = {
    auth: PropTypes.object.isRequired
};
// Mapeia o estado do Redux para as propriedades do componente
const mapStateToProps = state => ({
    auth: state.auth
});
// Conecta o componente ao Redux
export default connect(mapStateToProps)(PrivateRoute);
```

CAMINHO: client/src/utils/setAuthToken.js
ARQUIVO: setAuthToken.js

```
import axios from 'axios';
const setAuthToken = (token) => {
    if (token) {
        // Aplica token de autorização a cada requisição se estiver logado
        axios.defaults.headers.common['Authorization'] = token;
    } else {
        // Exclui o cabeçalho de usuário
        delete axios.defaults.headers.common['Authorization'];
    }
};
export default setAuthToken;
```

CAMINHO: client/src/reducers/authReducer.js
ARQUIVO: authReducer.js

```
import axios from 'axios';
const setAuthToken = (token) => {
    if (token) {
        // Aplica token de autorização a cada requisição se estiver logado
        axios.defaults.headers.common['Authorization'] = token;
    } else {
        // Exclui o cabeçalho de usuário
        delete axios.defaults.headers.common['Authorization'];
    }
};
export default setAuthToken;
```

CAMINHO: client/src/reducers/errorReducer.js
ARQUIVO: errorReducer.js

```
import { GET_ERRORS } from "../actions/types";

// Estado inicial do reducer
const initialState = {};

// Reducer que manipula as ações relacionadas a erros
export default function(state = initialState, action) {
    switch (action.type) {
        case GET_ERRORS:
            // Atualiza o estado com os erros recebidos na ação
            return action.payload;
        default:
            // Retorna o estado atual se a ação não for reconhecida
```

```
            return state;
    }
}
```

CAMINHO: client/src/reducers/index.js
ARQUIVO: index.js

```
import { combineReducers } from "redux";
import authReducer from "./authReducer";
import errorReducer from "./errorReducer";

// Combina os reducers em um único rootReducer
export default combineReducers({
    auth: authReducer,     // Reducer para o estado de autenticação do usuário
    errors: errorReducer   // Reducer para o estado de erros
});
```

CAMINHO: client/src/actions/authActions.js
ARQUIVO: authActions.js

```
import axios from "axios";
import setAuthToken from "../utils/setAuthToken";
import jwt_decode from "jwt-decode";
import {
    GET_ERRORS,
    SET_CURRENT_USER,
    USER_LOADING
} from "./types";

// Registra Usuário
export const registerUser = (userData, history) => dispatch => {
    axios
        .post("/api/users/register", userData)
        .then(res => history.push("/login")) /// redireciona para o login em caso de
registro bem-sucedido
        .catch(err =>
            dispatch({
                type: GET_ERRORS,
                payload: err.response.data
            })
        );
};

// Login - obter token do usuário
export const loginUser = userData => dispatch => {
    axios
        .post("/api/users/login", userData)
        .then(res => {
```

```
            // Salva no localStorage
            // Define o token no localStorage
            const { token } = res.data;
            localStorage.setItem("jwtToken", token);
            // Define o token no cabeçalho de autenticação
            setAuthToken(token);
            // Decodifica o token para obter os dados do usuário
            const decoded = jwt_decode(token);
            // Inclui o email no objeto do usuário
            decoded.email = userData.email;
            // Define o usuário atual
            dispatch(setCurrentUser(decoded));
        })
        .catch(err =>
            dispatch({
                type: GET_ERRORS,
                payload: err.response.data
            })
        );
};

// Define usuário logado
export const setCurrentUser = decoded => {
    return {
        type: SET_CURRENT_USER,
        payload: decoded
    };
};

// Carregamento do usuário
export const setUserLoading = () => {
    return {
        type: USER_LOADING
    };
};

// Desconectar usuário
export const logoutUser = () => dispatch => {
    // Remove o token do localStorage
    localStorage.removeItem("jwtToken");
    // Remove o cabeçalho de usuário para futuras requisições
    setAuthToken(false);
    // Define o usuário atual como um objeto vazio {} que definirá isAuthenticated
como false
    dispatch(setCurrentUser({}));
};

// Atualizar Preferências do Usuário
export const updateUserPreferences = (updatedPreferences, history) => (dispatch)
=> {
```

```
    dispatch(setUserLoading()); // Indica que a atualização está em andamento
    axios
        .put("/api/users/preferences", updatedPreferences)
        .then((res) => {
            // Supondo que as preferências foram atualizadas com sucesso
            console.log(res)
            // Redireciona para uma página de sucesso
            history.push("/products");
        })
        .catch((err) =>
            dispatch({
                type: GET_ERRORS,
                payload: err.response.data,
            })
        );
};
```

CAMINHO: client/src/actions/types.js
ARQUIVO: types.js

```
// Definição de constantes para ações Redux
export const GET_ERRORS = "GET_ERRORS";          // Obtém erros
export const USER_LOADING = "USER_LOADING";      // Carregamento do usuário
export const SET_CURRENT_USER = "SET_CURRENT_USER";  // Define o
usuário atual
```

CAMINHO: ./README.md
ARQUIVO: README.md

Vagou Aqui App
Bem-vindo ao Vagou Aqui App, uma aplicação web desenvolvida para auxiliar na busca e recomendação de anúncios imobiliários.

Visão Geral
Este projeto é uma aplicação de página única (SPA) que utiliza a stack MERN (MongoDB, Express.js, React.js, Node.js) para criar uma plataforma para busca e recomendação de anúncios imobiliários.

Estrutura do Diretório
A estrutura do diretório é organizada em duas partes principais: client (lado do cliente) e server (lado do servidor). Aqui está uma visão geral da estrutura do diretório:

```
|--- client
|--- server
|--- LICENSE.md
```

|--- README.md

Para mais detalhes sobre a estrutura do diretório, consulte o README.md nos diretórios client e server.

Configuração e Execução
Siga as instruções abaixo para configurar e executar a aplicação:

Instalação de Dependências:

No diretório client do projeto, execute npm install para instalar as dependências do cliente.
Em seguida, vá para o diretório server e execute npm install para instalar as dependências do servidor.
Configuração do Banco de Dados:

Crie um banco de dados com o o nome VagouAqui no mongoDB

Crie um arquivo .env no diretório server e preencha-o com as informações necessárias. Exemplo:

MONGODB_URI=
PORT=
LAST_SCRAPING_DATE=
Insira a string de conexão na variável MONGODB_URI=.

Insira o valor da porta do sevidor na variável PORT (sugestão 5000)

LAST_SCRAPING_DATE fica vazio porque será preenchido quando os anúncios forem raspados.

Execução:

No diretório server do projeto, execute npm run server para iniciar o servidor da aplicação.
No diretório client do projeto, execute npm run client para iniciar o cliente da aplicação.
Acesse http://localhost:3000/ para acessar a aplicação
Rotas do Servidor
Anúncios
GET /api/ads/all: Obtém todos os anúncios.
GET /api/ads/lastScrapingDate: Obtém a data da última extração.
Recomendações
GET /api/recommendations/all: Obtém recomendações de anúncios baseadas em conteúdo para um usuário autenticado.
Usuários
POST /api/users/register: Registra um novo usuário.
POST /api/users/login: Loga um usuário e retorna um token JWT.
PUT /api/users/preferences: Atualiza as preferências do usuário autenticado.
GET /api/users/preferences: Obtém as preferências do usuário autenticado.

GET /api/users/me: Obtém as informações do usuário autenticado.
DELETE /api/users/delete: Exclui a conta do usuário autenticado.
Scripts Disponíveis para o cliente
npm run dev: Inicia o servidor e o cliente em modo de desenvolvimento.
npm run build: Compila a aplicação para produção.
npm start: Inicia o servidor em modo de produção.
npm run client: Inicia o servidor.
Licença
Este projeto é licenciado sob a Licença MIT - consulte o arquivo LICENSE.md para obter detalhes.

Autor
Jose Ribamar Marcal Martins Junior

CAMINHO: ./LICENSE.md
ARQUIVO: LICENSE.md

CAMINHO: client/package.json
ARQUIVO: package.json

```
{
  "name": "vagou-aqui",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "axios": "^0.18.0",
    "classnames": "^2.2.6",
    "is-empty": "^1.2.0",
    "jwt-decode": "^2.2.0",
    "prop-types": "^15.8.1",
    "react": "^16.14.0",
    "react-dom": "^16.14.0",
    "react-image-lightbox": "^5.1.4",
    "react-redux": "^5.1.1",
    "react-responsive-carousel": "^3.2.23",
    "react-router-dom": "^4.3.1",
    "react-scripts": "2.1.1",
    "react-select": "^5.7.4",
    "redux": "^4.0.1",
    "redux-thunk": "^2.3.0",
    "web-vitals": "^3.4.0"
  },
  "scripts": {
    "client": "react-scripts --openssl-legacy-provider start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "proxy": "http://localhost:5000",
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": [
    ">0.2%",
    "not dead",
    "not ie <= 11",
    "not op_mini all"
  ]
}
```

CAMINHO: server/package.json
ARQUIVO: package.json

```
{
  "name": "vagou-aqui-app",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "client-install": "npm install --prefix client",
    "start": "node server.js",
    "server": "node server.js"
  },
  "keywords": [
    "mongodb",
    "mongoose",
    "react",
    "express",
    "nodejs",
    "ufsc"
  ],
  "author": "Jose Ribamar Marcal Martins Junior",
  "license": "MIT",
  "dependencies": {
    "axios": "^1.4.0",
    "axios-rate-limit": "^1.3.0",
    "axios-retry": "^3.8.0",
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.2",
    "cheerio": "^1.0.0-rc.12",
    "concurrently": "^8.2.0",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "is-empty": "^1.2.0",
    "jsonwebtoken": "^9.0.1",
    "lodash": "^4.17.21",
    "moment-timezone": "^0.5.43",
    "mongodb": "^5.7.0",
    "mongoose": "^7.4.2",
    "node-cron": "^3.0.2",
    "node-fetch": "^3.3.2",
    "node-schedule": "^2.1.1",
    "passport": "^0.6.0",
    "passport-jwt": "^4.0.1",
    "validator": "^13.11.0"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

'VAGOU AQUI': DESENVOLVIMENTO DE UM SISTEMA WEB DE AGREGAÇÃO DE VAGAS DE MORADIA VOLTADO PARA A COMUNIDADE ACADÊMICA DA UFSC

José R. M. M. Junior[1]

[1]Departamento de Informática e Estatística – INE - Universidade Federal de Santa Catarina - UFSC

CEP 88040-370 – Santa Catarina – SC – Brazil

j.r.junior@grad.ufsc.br

Abstract. *The "VagouAqui" web application represents a comprehensive solution to the daunting task of locating rental housing in Florianópolis. This innovative platform operates by utilizing scrapers to extract housing ads from five distinct sources, streamlining the otherwise challenging search process. Its architectural framework follows a client-server model while implementing the Model-View-Controller (MVC) programming pattern. The user-centric front-end, powered by React.js, prioritizes an intuitive and seamless interface, enhancing user experience. Meanwhile, the crucial back-end, developed on Node.js, plays a pivotal role in amalgamating and organizing data from various scrapers, thereby simplifying the retrieval of information from multiple sources. Additionally, the application boasts a sophisticated recommendation function that offers personalized suggestions based on individual user preferences, enhancing the overall user journey. It's worth noting that while the initial plan was to deploy the application on the AWS cloud infrastructure, this specific goal was not achieved during the development phase. Nonetheless, the potential for future integration into AWS remains a valuable prospect and is recommended for further enhancements and scalability of the platform.*

Resumo. *Este trabalho apresenta o registro do desenvolvimento da aplicação web "VagouAqui", uma solução abrangente para a busca desafiadora por moradia para aluguel em Florianópolis. Utilizando scrapers para coletar dados de cinco fontes distintas, a aplicação permite aos usuários se cadastrarem e receberem anúncios personalizados com base em suas preferências. Com uma arquitetura cliente-servidor e a estruturação MVC, o front-end em React.js oferece uma interface amigável, enquanto o back-end em Node.js integra os dados dos scrapers, simplificando a obtenção de informações. A função de recomendação oferece sugestões personalizadas, embora a implantação na nuvem AWS, inicialmente planejada, não tenha sido realizada durante o desenvolvimento, permanecendo como uma sugestão valiosa para futuras melhorias..*

1. Introdução

Este artigo discute a crescente demanda por moradia entre os estudantes universitários da Universidade Federal de Santa Catarina (UFSC) em Florianópolis, uma cidade classificada como uma das melhores do Brasil. A falta de áreas disponíveis e o aumento das ocupações informais têm desafiado os estudantes na busca por moradia acessível e bem localizada. A escassez de espaços, aliada à dispersão dos anúncios de aluguel em várias plataformas, à falta de verificação da procedência dos anúncios e à ausência de feedback dos usuários comprometem a confiabilidade e a experiência na busca por moradia. Diante desse cenário, propõe-se a criação de uma plataforma digital confiável que agregue anúncios de aluguel em Florianópolis e na área metropolitana, permitindo interações entre os usuários e feedback sobre os anúncios. Esta solução não apenas busca atender à demanda por moradia acessível para estudantes da UFSC, mas também alinha-se à importância crescente das plataformas digitais na democratização do acesso à informação e transparência no mercado imobiliário, contribuindo para a busca de soluções inovadoras para problemas sociais.

2. Objetivos

O objetivo geral é criar uma plataforma que reúna anúncios de aluguel de 5 plataformas: Classificados UFSC, IBAGY imóveis, VivaReal Imóveis, MGF Imóveis e WebQuartos. Além disso, implementar uma função de recomendação de anúncios de aluguel para os usuários cadastrados na aplicação que leve em consideração as preferências individuais cadastradas no perfil do usuário.

Os objetivos específicos abrangem a expansão do conhecimento em modelos de web scraping e recomendação de anúncios. Isso inclui a seleção cuidadosa de modelos adequados para a coleta de anúncios de aluguel e para recomendação personalizada de anúncios e usuários afins. O desenvolvimento da aplicação web, contemplando tanto o backend quanto o frontend, foca na agregação de anúncios de aluguel em Florianópolis, com a utilização dos serviços da AWS para hospedagem, garantindo uma infraestrutura confiável.

3. Tecnologias Utilizadas

A stack MERN (MongoDB, Express, React e NodeJS) foi escolhida para para o desenvolvimento do aplicativo "VagouAqui". Essa decisão foi motivada pela popularidade do NodeJS na criação de APIs Rest e pela flexibilidade do Express para o gerenciamento dessas APIs. Além disso, o React foi escolhido devido à sua facilidade na criação de interfaces de usuário (UIs), enquanto o MongoDB proporciona uma forma simples de armazenamento de dados. A aplicação também faz uso de outras tecnologias para garantir uma experiência de usuário eficiente e escalável. No lado do cliente, o React Redux é empregado para o gerenciamento de estado, o React Router é utilizado para a navegação entre diferentes seções do aplicativo sem a necessidade de recarregar a página completa, e a biblioteca Axios garante a eficiência das requisições HTTP entre o frontend e o backend. No lado do servidor, a segurança é assegurada com o uso de JSON Web Tokens (JWT) para autenticação, enquanto as senhas são criptografadas utilizando Bcrypt. Adicionalmente, a aplicação inclui funcionalidades de raspagem de conteúdo web com a ajuda da biblioteca Cheerio e a capacidade de agendamento de tarefas utilizando as bibliotecas Cron e Schedule. A Figura 1. a seguir ilustra a arquitetura.

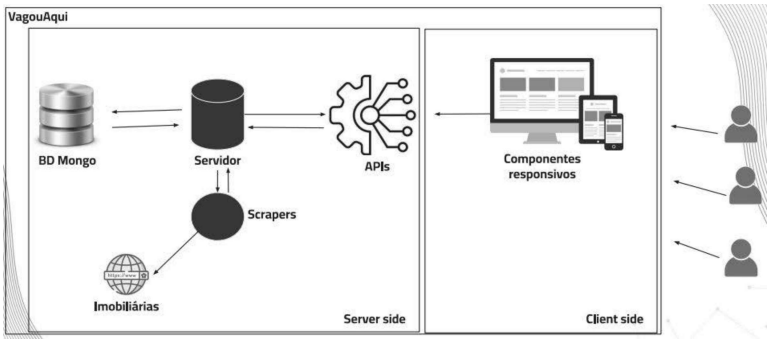


Figura 1. Arquitetura da aplicação 'VagouAqui'

4. Rotina de coleta de dados

Todos os scrapers seguem uma rotina similar. Inicialmente, configurações e bibliotecas são preparadas para permitir a comunicação com o site (por exemplo, axios para realizar solicitações HTTP e cheerio para analisar o conteúdo HTML das páginas). O código então acessa páginas específicas do site buscando por anúncios imobiliários. Ao encontrar esses anúncios, extrai os links correspondentes para cada um deles. Em seguida, para cada link coletado, o código faz novas requisições ao site para obter informações detalhadas sobre o respectivo anúncio. Esses detalhes incluem aspectos como título, descrição, preço e localização. Há também uma parte do código dedicada a extrair informações específicas, como o bairro a partir do endereço fornecido nos anúncios. Depois disso, são utilizados duas funções auxiliares para extração do telefone de contato e dos links das imagens. Ao final, todas as informações coletadas são estruturadas e organizadas em um objeto JSON para formar um conjunto de dados que representam os anúncios imobiliários.

5. Função de recomendação

O código gera recomendações de anúncios baseadas nas preferências do usuário. Ele acessa os anúncios no banco de dados e ajusta a quantidade considerada dependendo do total disponível. Em seguida, analisa as preferências do usuário, como orçamento e outras características desejadas. Iterando sobre os anúncios, compara suas características com as preferências do usuário, atribuindo pontuações. Ao final, retorna uma lista ordenada dos anúncios mais alinhados com as preferências, dentro de um tamanho pré-determinado, oferecendo recomendações personalizadas para o usuário.

6. Componentes React

Na aplicação 'VagouAqui' são utilizados diferentes componentes React para oferecer funcionalidades específicas. Alguns desses componentes são responsáveis por exibir conteúdos dinâmicos, como anúncios em uma grade paginada ou imagens em um carrossel, proporcionando uma experiência visual interativa. Outros componentes atuam como interfaces de interação, como o registro de novos usuários, permitindo que eles insiram suas informações pessoais e definam preferências específicas para uma melhor correspondência com os conteúdos disponíveis. Também há um componente para edição das preferências do usuário e um componente que é o perfil do usuário, no qual ele pode verificar as informações cadastradas e excluir a conta dele, caso deseje. Essa variedade de componentes atende a diferentes necessidades da aplicação, desde a apresentação de informações até a interação e personalização da experiência do usuário.

7. Referências

BHARDWAJ, Sushil; JAIN, Leena; JAIN, Sandeep. Cloud computing: A study of infrastructure as a service (IAAS). International Journal of engineering and information Technology, v. 2, n. 1, p. 60-63, 2010.

MATHEW, Sajee; VARIA, J. Overview of amazon web services. Amazon Whitepapers, v. 105, p. 1-22, 2014.

AMAZON WEB SERVICES. Amazon EC2. Disponível em: <https://aws.amazon.com/ec2/>. Acesso em: 18 abr. 2023.

PATTERSON, Scott. Learn AWS Serverless Computing: A Beginner's Guide to Using AWS Lambda, Amazon API Gateway, and Services from Amazon Web Services. Packt Publishing Ltd, 2019.

AMAZON WEB SERVICES. Amazon Lambda AWS. Disponível em: <https://aws.amazon.com/pt/lambda/>. Acesso em: 18 abr. 2023.

AMAZON WEB SERVICES. Provisionamento de Infraestrutura como código - AWS. Disponível em: <https://aws.amazon.com/pt/cloudformation/>. Acesso em: 18 abr. 2023.

SAHU, Saurabh et al. Sentimental Analysis on Web Scraping Using Machine Learning Method. Journal of Information and Computational Science (JOICS), ISSN, p. 1548-7741.

The industry standard for working with HTML in JAvaScript. Cheerio. Disponível em <https://cheerio.js.org/>. Acesso em: 20 de outubro de 2023.

DA MOTTA, Claudia Lage Rebello et al. Sistemas de recomendação. 2012.

CHAFFER, Jonathan. Learning jQuery. Packt Publishing Ltd, 2013.

Wong, C.-I & Wong, Kin Yeung & Ng, K.-W & Fan, W. & Yeung, Alan. (2014). Design of a crawler for online social networks analysis. WSEAS Transactions on Communications. 13. 263-274.

LUCIANO, Josué; ALVES, Wallison Joel Barberá. Padrão de arquitetura MVC: Model-view-controller. EPeQ Fafibe, v. 1, n. 3a, p. 102-107, 2017.

CHAUHAN, Anjali. A review on various aspects of MongoDB databases. International Journal of Engineering Research & Technology (IJERT), v. 8, n. 05, p. 90-92, 2019.

MARDAN, Azat. Express. js Guide: The Comprehensive Book on Express. js. Azat Mardan, 2014.

CANTELON, Mike et al. Node. js in Action. Greenwich: Manning, 2014.

RFC 7519 - JSON Web Token. RFC. Disponível em: <https://datatracker.ietf.org/doc/html/rfc7519>. Acesso em: 20 de outubro de 2023.