UNIVERSIDADE FEDERAL DE SANTA CATARINA

Aplicativo móvel para busca de imóveis para universitários

Matheus Andrade Alves

Florianópolis - SC

2021/2

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

Aplicativo móvel para busca de imóveis para universitários

Matheus Andrade Alves

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

Florianópolis - SC
2021/2

MATHEUS ANDRADE ALVES

Aplicativo móvel para busca de imóveis para universitários

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

Orientador:

_____

Frank Augusto Siqueira

Banca examinadora:

_____

Alex Sandro Roschildt Pinto

_____

Jean Carlo Rossa Hauck

SUMÁRIO

RESUMO

Neste trabalho será abordado o tema de busca de imóveis por universitários. Atualmente existem alguns sites que oferecem serviços para locadores anunciarem seus imóveis especificamente para universitários. Porém, estes sites não possuem uma usabilidade boa e não fazem um bom uso de um sistema de filtros levando em conta as necessidades de um estudante universitário que deseja morar perto de sua faculdade e queira encontrar uma acomodação que encaixe no seu orçamento e que tenha as características que atendem às suas necessidades. O objetivo deste trabalho é desenvolver um aplicativo móvel nas plataformas Android e iOS para alunos de universidades procurarem acomodações, por exemplo imóveis, repúblicas ou quartos por meio de um mapa, utilizando filtros específicos para universitários. Para isso são utilizados métodos de engenharia de software para definição do produto e da arquitetura do software. Ao final do trabalho, é realizada uma pesquisa com os alunos para validar o aplicativo, sua usabilidade e funcionalidades, a qual retorna resultados positivos e valida os objetivos do trabalho.

Palavras chave: Flutter, imóveis, universitários, aplicativo

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Atualmente não existem opções de aplicativos de busca de imóveis para aluguel especificamente voltados às necessidades específicas de alunos universitários, segundo a pesquisa realizada por este trabalho. O aluno que deseja procurar um imóvel perto da UFSC tem a opção de procurar um imóvel em um site ou aplicativo de corretoras de imóveis, as quais nem sempre têm em mente este público alvo e não possuem ferramentas específicas para filtrar os imóveis de acordo com as necessidades de alunos de universidades. Além dos sites de corretoras, existe o site de classificados da UFSC, disponível em https://classificados.inf.ufsc.br, onde há uma seção de classificados para imóveis, no qual aparentemente a maioria dos alunos procura por um lugar para morar perto da universidade. Porém, este site peca na usabilidade e não possibilita nenhum tipo de filtro nas buscas, dificultando bastante a procura por um imóvel.

Tendo este problema em mente, este trabalho propõe o desenvolvimento de um aplicativo móvel específico para alunos universitários que desejam procurar uma moradia perto da universidade. Para isso, neste trabalho será pesquisado o estado da arte nos atuais aplicativos de imóveis para entender quais funcionalidades são mais úteis e desejadas pelos usuários, e também serão investigados os atuais aplicativos específicos para universitários para entender quais são os itens mais procurados pelos alunos ao procurar por um imóvel.

Com estes dados será desenvolvido um aplicativo móvel para *Android* e *iOS* que terá os anúncios de imóveis disponibilizados em um mapa, onde os imóveis serão representados por marcadores. Os usuários que procuram um imóvel poderão inserir algumas informações pessoais como: curso, centro, meio de transporte que será utilizado para locomoção até a faculdade, entre outros, para ajudar na busca de um imóvel. Além disso, serão desenvolvidos filtros específicos para o público alvo baseado nos estudos anteriores, em específico um filtro para a distância entre o imóvel e o centro de estudos do aluno, de modo que o aluno possa filtrar anúncios por tempo máximo de deslocamento entre o imóvel e o centro de estudos.

Para otimizar os anúncios, o aplicativo irá coletar dados dos usuários, como por exemplo em quais anúncios tal usuário clicou e por quanto tempo ficou neles, e junto com os dados fornecidos pelo usuário serão geradas estatísticas que fornecerão dados para os locadores.

Espera-se que com o aplicativo, os alunos consigam procurar com facilidade imóveis para alugar perto da sua universidade e que atendam às suas necessidades para estudos. Para validar esse objetivo será realizada uma avaliação com os usuários ao fim do projeto a fim de receber *feedback* sobre o aplicativo, suas funcionalidades e usabilidade.

1.1 Objetivo Geral

Desenvolver um aplicativo móvel nas plataformas *Android* e *iOS* para alunos de universidades procurarem acomodações, por exemplo imóveis, repúblicas ou quartos por meio de um mapa, utilizando filtros específicos para universitários.

1.2 Objetivos Específicos

Como objetivos específicos foram listados os quatro principais:

1. Realizar uma pesquisa sobre os atuais aplicativos móveis e sites de imóveis para público geral e compará-los com os aplicativos móveis e sites exclusivos para universitários;
2. Desenvolver casos de uso e esboços de interfaces gráficas para o aplicativo móvel;
3. Desenvolver o aplicativo móvel e publicar nas lojas de aplicativos *Play Store* e *App Store;*
4. Realizar uma avaliação do aplicativo por meio de um formulário eletrônico com o público alvo

1.3 Metodologia

Metodologia de desenvolvimento divide o projeto em diversos estágios e tem como objetivo organizar o processo de produção de um software para o melhor aproveitamento do tempo disponível. Existem diversos tipos de metodologias de desenvolvimento, como por exemplo, cascata, espiral, e ágil.

Para o desenvolvimento deste projeto, foi escolhido o desenvolvimento ágil, mais especificamente *Scrum*. O desenvolvimento ágil de software refere-se a um grupo de metodologias de desenvolvimento de software baseadas no desenvolvimento iterativo, onde os requisitos e soluções evoluem por meio da colaboração entre equipes auto-organizadas. Metodologias ágeis geralmente

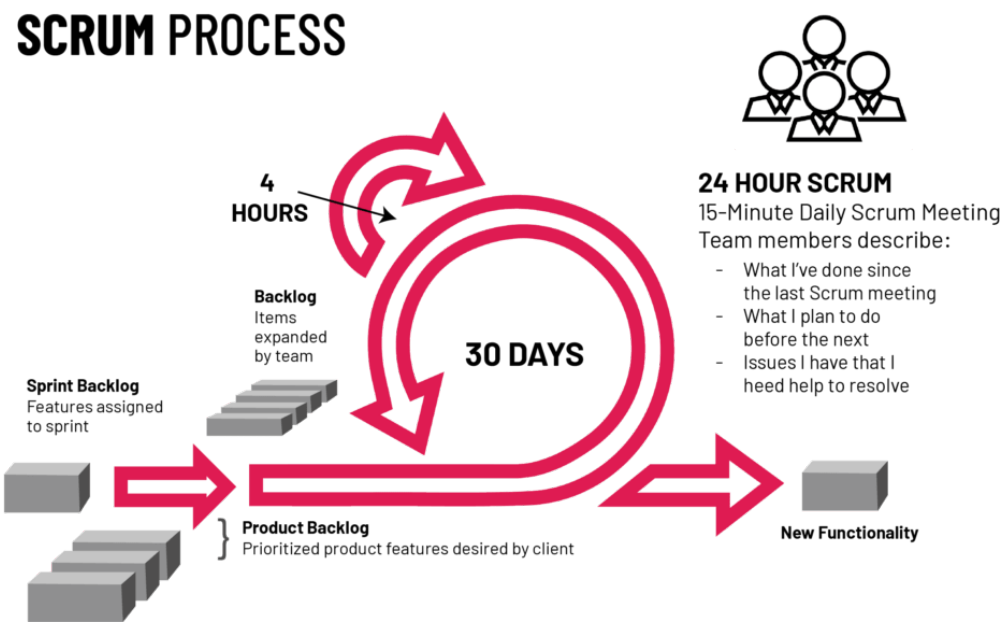
promovem um processo disciplinado de gerenciamento de projetos que incentiva a inspeção e adaptação frequentes, um conjunto de melhores práticas de engenharia destinadas a permitir a entrega rápida de software de alta qualidade, e uma abordagem de negócios que alinha o desenvolvimento com as necessidades do cliente e os objetivos da empresa (Agile Alliance, 2021)

Scrum é um framework de processos para desenvolvimento ágil mais usado atualmente. A base de funcionamento do Scrum é a divisão do processo de produção em iterações de pequenos períodos de tempo, os quais são conhecidos como Sprints, sendo que o tamanho mais comum para um Sprint é de duas semanas (Agile Alliance, 2021).

Todas as funcionalidades que precisam ser desenvolvidas no projeto ficam em uma lista chamada *Product Backlog* e são definidas pelo *Product Owner*. As tarefas que serão desenvolvidas em cada *Sprint* são definidas numa reunião chamada de *Sprint Planning Meeting*, na qual o *Product Owner* prioriza funcionalidades que ainda estão no *Product Backlog* enquanto a equipe de desenvolvimento seleciona atividades que terão a capacidade de terminar dentro daquele Sprint que está para iniciar, movendo cada item do *Product Backlog* para o *Sprint Backlog*.

Cada dia dentro do período da *Sprint*, a equipe faz uma reunião chamada de *Daily Scrum*, tendo como objetivo repassar o que foi feito no dia anterior, apontar problemas ocorridos e detalhar o que será feito no dia que está para começar. Ao final da Sprint, a equipe de desenvolvimento apresenta as funcionalidades que foram desenvolvidas numa reunião chamada *Sprint Review Meeting* e, por fim, é feito uma *Sprint Retrospective* para apontar o que deu certo, o que não deu certo e o que pode ser melhorado para o próximo *Sprint*. Após isso, reinicia-se o ciclo, conforme mostra a Figura 1.

Figura 1 - Funcionamento do *Scrum*



Fonte: https://www.cprime.com/resources/what-is-agile-what-is-scrum/

No contexto do projeto, o *Product Owner* é o orientador do projeto e o time de desenvolvimento é o orientando. As *Sprints* foram definidas com duração de 2 semanas, onde ao final dessas 2 semanas, é feita uma *Sprint Review* onde é apresentado ao orientador o que foi feito. Logo após é feito o *Sprint Planning*, onde o time inteiro define quais serão as atividades para as próximas duas semanas, reiniciando o ciclo. Reuniões diárias e retrospectivas não serão realizadas, pois o time de desenvolvimento possui somente uma pessoa.

1.4 Organização do Trabalho

Este trabalho está organizado em 7 capítulos. O capítulo 2 tem como objetivo apresentar os conceitos fundamentais apresentados no projeto, com o intuito de apresentar uma visão geral sobre tecnologias utilizadas no desenvolvimento do projeto. O capítulo 3 descreve o estado da arte dos serviços semelhantes existentes atualmente relacionados ao projeto. O capítulo 4 apresenta a proposta da solução, descrevendo a concepção do aplicativo e como ele deve funcionar. No capítulo 5 serão demonstrados o processo de desenvolvimento do aplicativo e suas funcionalidades. O capítulo 6 apresenta uma pesquisa de usabilidade do aplicativo, para identificar seus pontos positivos e negativos. O capítulo 7 apresenta os

resultados obtidos com a criação e testes do aplicativo e considerações finais. E por fim, o capítulo 8 apresenta os possíveis trabalhos futuros que podem agregar para o amadurecimento do aplicativo.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos sobre aplicativos móveis e tecnologias utilizadas para o desenvolvimento do projeto.

2.1 Aplicativo Móvel

Um aplicativo móvel, mais comumente referido como um aplicativo, é um tipo de software projetado para ser executado em um dispositivo móvel, como um smartphone ou tablet. Os aplicativos móveis frequentemente servem para fornecer aos usuários serviços semelhantes aos acessados em computadores pessoais. Os aplicativos são geralmente unidades de software pequenas e individuais com funções limitadas. Esse uso de software de aplicativo foi originalmente popularizado pela Apple e sua App Store, que oferece milhares de aplicativos para iPhone e iPad (MROCZKOWSKA, 2021).

Por causa dos recursos de hardware limitados dos primeiros dispositivos móveis, os aplicativos móveis evitavam a multifuncionalidade. No entanto, mesmo que os dispositivos usados hoje sejam muito mais sofisticados, os aplicativos móveis permanecem estritamente funcionais. É assim que os proprietários de aplicativos móveis permitem que os consumidores escolham exatamente as funções que seus dispositivos devem ter.

Os aplicativos são divididos em três grandes categorias: aplicativos nativos, aplicativos web e aplicativos híbridos. Os aplicativos nativos são desenvolvidos para um sistema operacional móvel específico, geralmente iOS ou Android. Os aplicativos nativos desfrutam de melhor desempenho e de uma interface de usuário mais ajustada (MROCZKOWSKA, 2021).

Aplicativos web são desenvolvidos em HTML5 ou CSS e requerem memória mínima do dispositivo, pois são executados por meio de um navegador. O usuário é redirecionado para uma página da web específica e todas as informações são salvas em um banco de dados baseado no servidor. Os aplicativos web requerem uma conexão estável para serem usados.

Aplicativos híbridos são desenvolvidos usando tecnologias da web, como JavaScript, CSS e HTML 5. São basicamente como aplicativos web disfarçados em um invólucro nativo. Os aplicativos híbridos são fáceis e rápidos de desenvolver, e há apenas uma única base de código para todas as plataformas. Isso diminui o

custo de manutenção e agiliza o processo de atualização. Por outro lado, os aplicativos híbridos podem carecer de velocidade, desempenho e o aplicativo pode não ter a mesma aparência em duas ou mais plataformas (MROCZKOWSKA, 2021).

2.2 Flutter

Flutter é um framework de desenvolvimento híbrido que visa possibilitar o desenvolvimento de aplicativos móveis de alto desempenho utilizando a linguagem de programação Dart. Flutter foi lançado publicamente em 2016 pelo Google. Não apenas os aplicativos Flutter podem ser executados no Android e iOS, mas também o Fuschia, o sistema operacional de próxima geração do Google, o qual usará Flutter para desenvolvimento de seus aplicativos (GOOGLE, Inside Flutter, 2021).

Em vez de utilizar visualizações da web, como mencionado em um item anterior, ou depender dos artefatos visuais do sistema operacional do smartphone, o Flutter renderiza todos os componentes de visualização usando seu próprio mecanismo de renderização de alto desempenho. Desta forma, ele consegue oferecer a possibilidade de construir aplicativos que apresentam desempenho similar aos aplicativos nativos. Em termos de arquitetura, o código *C/C++* da *engine* é compilado com o NDK (*Native Development Kit*) do Android e com LLVM no iOS, e qualquer código Dart é compilado AOT (*Ahead-of-time*) em código nativo no momento do empacotamento do aplicativo (GOOGLE, Flutter architectural overview, 2021).

Flutter é um framework de interface reativa, no qual o desenvolvedor define um mapeamento do estado do aplicativo para o estado da interface, e o framework assume a tarefa de atualizar a interface em tempo de execução quando o estado do aplicativo muda. Este modelo é inspirado no trabalho que veio do Facebook para seu próprio framework React.

O Flutter oferece suporte para *hot reload* com monitoração de estado durante o desenvolvimento, o que é considerado um fator importante para impulsionar o ciclo de desenvolvimento. O recurso *hot reload* do Flutter ajuda o desenvolvedor a experimentar, construir interfaces de usuário, adicionar recursos e corrigir bugs de forma rápida e fácil. O *hot reload* funciona injetando arquivos de código-fonte atualizados na *Dart Virtual Machine* (VM) em execução. Depois que a VM atualiza as classes com as novas versões de campos e funções, a estrutura Flutter

reconstrói automaticamente a árvore de *widgets*, permitindo que o desenvolvedor visualize rapidamente os efeitos de suas alterações sem precisar compilar novamente o aplicativo (Google, 2021).

2.2.1 Dart

No Flutter, todos os aplicativos são desenvolvidos em Dart. Dart é uma linguagem de programação desenvolvida e mantida pelo Google. É amplamente utilizada no Google e provou ter a capacidade de desenvolver aplicativos da web massivos, como o AdWords (Google, 2016).

Dart foi originalmente desenvolvido como um substituto e sucessor do JavaScript. Assim, ele implementa a maioria das características importantes do próximo padrão do JavaScript (ES7), como por exemplo as palavras-chave "async" e "await". No entanto, para atrair desenvolvedores que não estão familiarizados com JavaScript, o Dart tem uma sintaxe semelhante à linguagem Java.

De forma similar a outros sistemas que utilizam programação reativa, o Flutter atualiza a árvore de visualização a cada novo quadro. Esse comportamento leva a uma desvantagem de que muitos objetos, que podem viver por apenas um quadro, serão criados. O Dart, como uma linguagem de programação moderna, é otimizado para lidar com este cenário em nível de memória com a ajuda do *garbage collector*.

2.2.2 Widgets

Widgets são os elementos mais importantes no Flutter. Eles são construídos usando uma estrutura moderna que se inspira no *React*. A ideia central é que construa-se uma interface de usuário a partir de *widgets*. Os *widgets* descrevem a aparência de sua visualização de acordo com sua configuração e estado atual. Quando o estado de um *widget* muda, o *widget* reconstrói sua descrição, que o Flutter compara com a versão anterior a fim de determinar as mudanças mínimas necessárias na árvore de renderização subjacente para a transição de um estado para o próximo.

Figura 2 - Fórmula matemática para o layout da tela.

Fonte: https://flutter.dev/docs/development/data-and-backend/state-mgmt/declarative

No Flutter, os *widgets* são representados por classes imutáveis que são usadas para configurar uma árvore de objetos, conforme mostra a Figura 2. Esses *widgets* são usados para gerenciar uma árvore separada de objetos para *layout*, que é então usada para gerenciar uma árvore separada de objetos para composição. Flutter é, em sua essência, uma série de mecanismos para caminhar com eficiência pelas partes modificadas das árvores, convertendo árvores de objetos em árvores de objetos de nível inferior e propagando mudanças por essas árvores (Google, 2021).

Conforme mencionado, o Flutter enfatiza os *widgets* como uma unidade de composição. *Widgets* são os blocos de construção da interface do usuário de um aplicativo Flutter, e cada *widget* é uma declaração imutável de parte da interface do usuário. Os *widgets* formam uma hierarquia com base na composição, formando uma estrutura que vai até o *widget* raiz. Cada *widget* se aninha dentro de seu pai e pode receber contexto do pai.

2.3 REST API

Uma API é um conjunto de definições e protocolos para construir e integrar softwares. Uma API pode ser vista como um contrato entre um provedor de informações e um usuário de informações – estabelecendo o conteúdo exigido do consumidor (a chamada) e o conteúdo exigido pelo produtor (a resposta). Por exemplo, o design da API para um serviço meteorológico pode especificar que o usuário forneça um CEP e que o produtor responda com uma resposta em duas

partes, sendo a primeira a temperatura alta e a segunda a temperatura baixa (IBM, 2022).

Em outras palavras, se deseja interagir com um sistema para recuperar informações ou executar uma função, uma API o ajuda a comunicar o que deseja a esse sistema para que ele possa entender e atender à solicitação.

REST é um conjunto de restrições arquitetônicas, não um protocolo ou padrão. Os desenvolvedores de API podem implementar REST de várias maneiras.

Quando uma solicitação de cliente é feita por meio de uma API *RESTful*, ela transfere uma representação do estado do recurso para o solicitante ou *endpoint*. Esta informação, ou representação, pode ser entregue em vários formatos via *HTTP: JSON (Javascript Object Notation), HTML, XLT, Python,* texto simples, entre outros. *JSON* é o formato de arquivo mais popular para uso em APIs REST porque, apesar de seu nome, é independente de linguagem, bem como legível por humanos e máquinas.

Sendo assim, por exemplo, uma API REST pode ser utilizada por um aplicativo móvel para interagir com um serviço de listagem de imóveis, onde o serviço responderia com uma lista de imóveis e suas informações em um formato *JSON*.

3. ESTADO DA ARTE

Este capítulo tem como finalidade identificar o estado atual em que se encontram os aplicativos comerciais e não-comerciais relacionados a anúncios de imóveis, voltados a universitários ou não.

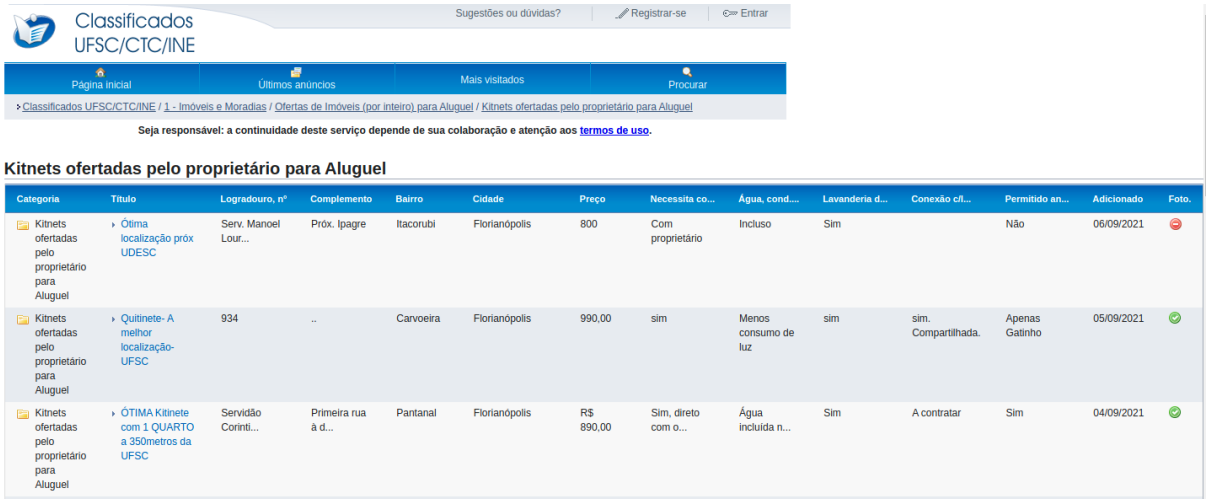
As ferramentas foram encontradas através de buscas pelo Google durante o mês de agosto de 2021, utilizando como palavras-chave: aluguel, ufsc, imóveis, Florianópolis, universidade.

Primeiramente foram selecionadas as ferramentas que tivessem alguma relação direta com a UFSC ou alguma outra universidade, após isto, foram selecionadas as ferramentas mais populares atualmente no mercado, que no caso são as mais destacadas nas pesquisas do *Google*.

3.1 Classificados UFSC

O Classificados UFSC é um serviço mantido pelo Departamento de Informática e Estatística da Universidade Federal de Santa Catarina, disponibilizado no endereço https://classificados.inf.ufsc.br/ . O serviço tem como objetivo a negociação ou troca de bens entre alunos, servidores e funcionários da universidade. Esse serviço é disponibilizado apenas no formato de uma página web, e dentro dele há uma área específica para anúncios de imóveis, conforme mostrado na Figura 3.

Figura 3 - Captura de tela das Kitnets ofertadas no website Classificados UFSC

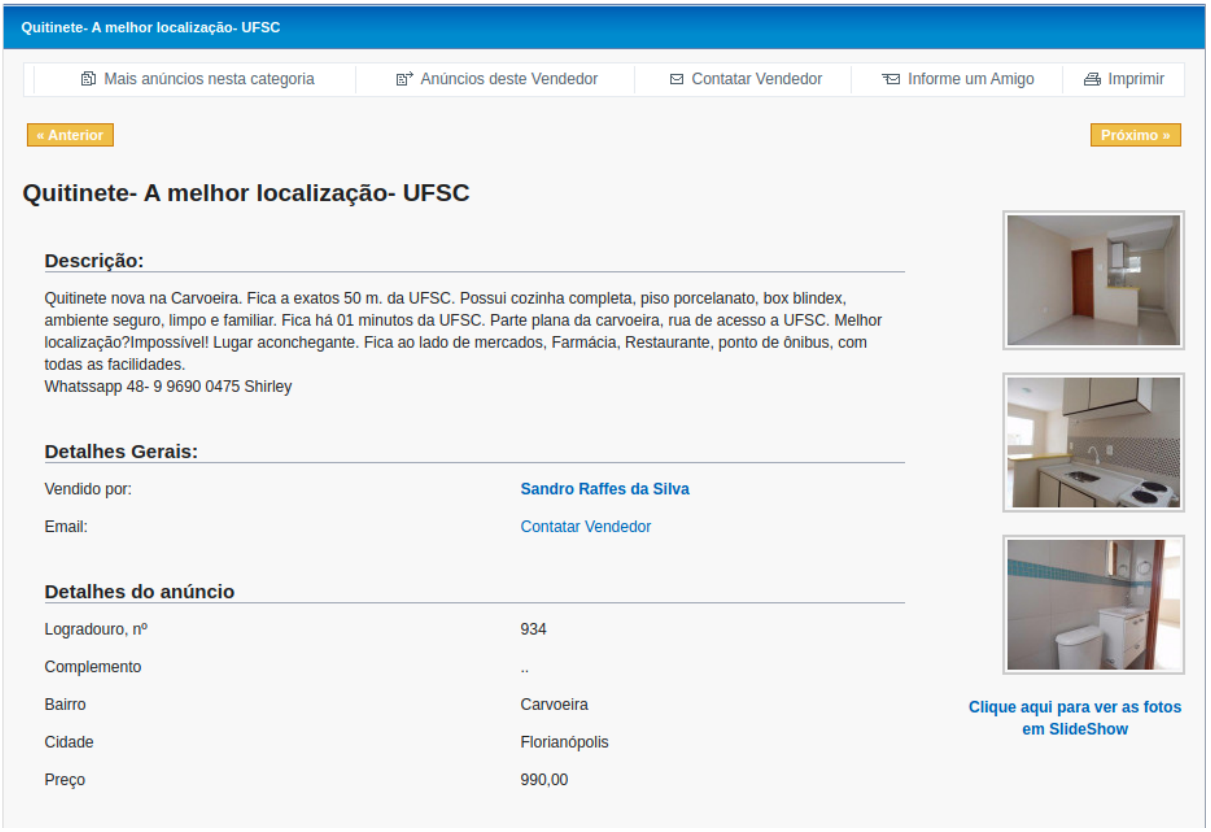


Categoria	Título	Logradouro, nº	Complemento	Bairro	Cidade	Preço	Necessita co...	Água, cond....	Lavanderia d...	Conexão c/i...	Permitido an...	Adicionado	Foto.
Kitnets ofertadas pelo proprietário para Aluguel	Ótima localização próx UDESC	Serv. Manoel Lour...	Próx. Ipagre	Itacorubi	Florianópolis	800	Com proprietário	Incluso	Sim		Não	06/09/2021	
Kitnets ofertadas pelo proprietário para Aluguel	Quitinete- A melhor localização- UFSC	934	..	Carvoeira	Florianópolis	990,00	sim	Menos consumo de luz	sim	sim. Compartilhada.	Apenas Gatinho	05/09/2021	
Kitnets ofertadas pelo proprietário para Aluguel	ÓTIMA Kitinete com 1 QUARTO a 350metros da UFSC	Servidão Corinti...	Primeira rua à d...	Pantanal	Florianópolis	R$ 890,00	Sim, direto com o...	Água incluída n...	Sim	A contratar	Sim	04/09/2021	

Fonte: http://classificados.inf.ufsc.br/ (2021)

O site apresenta uma interface antiga e de difícil utilização. Não é possível filtrar os anúncios de nenhuma forma, apenas é possível ordená-los ao clicar no cabeçalho da coluna. Porém, como os caracteres dos campos não são tratados, a ordem dos anúncios fica incorreta. Por exemplo, ao ordenar pelo campo “Preço”, como há alguns anúncios usando apenas números (800) e outros usando o símbolo da moeda também (R$ 800) a ordenação fica incorreta.

Figura 4 - Captura de tela de um anúncio de imóvel do website Classificados UFSC



Fonte: http://classificados.inf.ufsc.br/ (2021)

A Figura 4 mostra um anúncio de imóvel. O serviço permite adicionar no máximo três imagens, limitando a quantidade de informações que podem ser repassadas para o usuário do serviço.

3.2 Federal SC

Federal SC é um website de classificados para alunos da Universidade Federal de Santa Catarina, disponível em https://www.federalsc.com.br/. O website

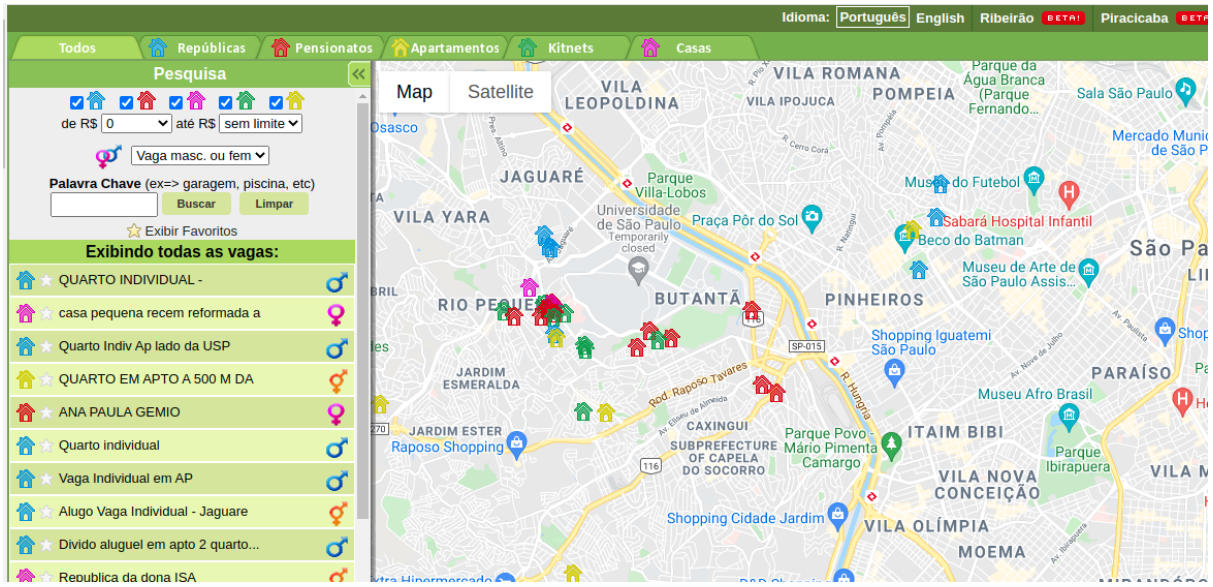
apresenta os anúncios no formato de carrossel e possui apenas filtros por localidade, no caso pelos bairros próximos da universidade, não sendo possível filtrar os imóveis por características como quantidade de quartos, preço, vaga de garagem, etc. A Figura 5 mostra um anúncio de imóvel.

Figura 5 - Captura de tela de um anúncio de imóvel do website Federal SC

Fonte: https://www.federalsc.com.br/ (2021)

3.3 Morar USP

Figura 6 - Captura de tela do mapa de imóveis do website Morar USP



Fonte: https://www.federalsc.com.br/ (2021)

Morar USP é um website, disponível em https://morarusp.com.br/, que foi desenvolvido pelos graduandos Eduardo Cauli, Victor Emanuel e Anderson Bravalheri dos cursos de Engenharia da Computação e Engenharia Elétrica da Universidade de São Paulo.

O sistema é voltado para o público universitário, sendo possível filtrar moradias por Repúblicas, Pensionatos, Apartamentos, Kitnets e Casas. Os imóveis são mostrados em um mapa, tendo uma cor específica para os tipos de moradias mencionados anteriormente, conforme mostra a Figura 6.

Os imóveis podem ser filtrados por faixa de preço e também se o local aceita apenas homens ou mulheres, ou ambos.

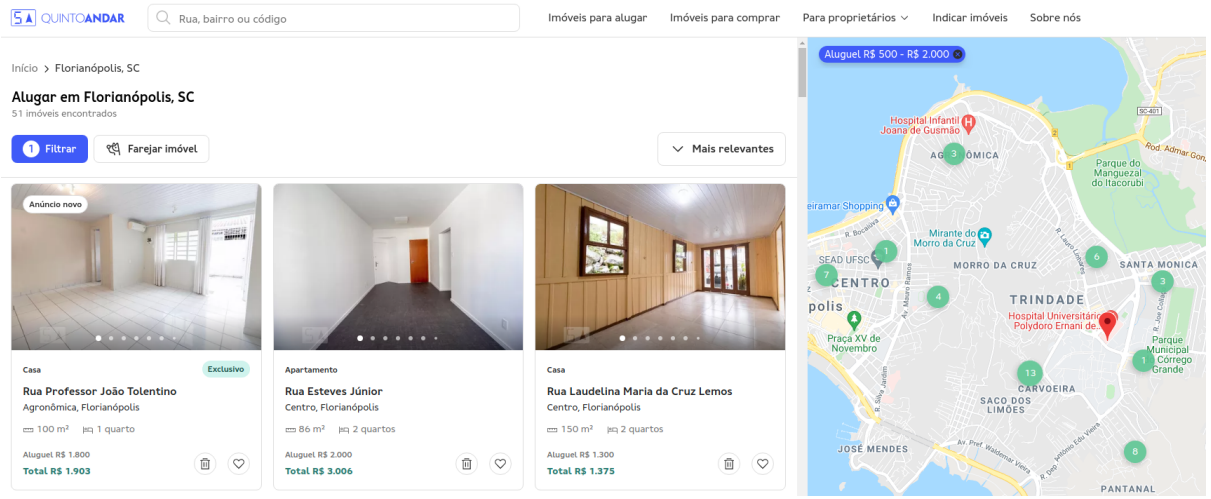
3.4 Quinto Andar

A QuintoAndar é uma startup brasileira de tecnologia com foco no aluguel e venda de imóveis. Na modalidade aluguel, a empresa administra o pagamento do aluguel ao proprietário, dispensando o locatário da apresentação de fiador, fiança ou caução. Na modalidade de venda, a empresa permite a negociação direta entre comprador e vendedor, e a partir daí cuida de todo o processo, incluindo *due*

diligence, que é a busca e análise prévia de informações, neste caso sobre a titularidade e financiamento bancário.

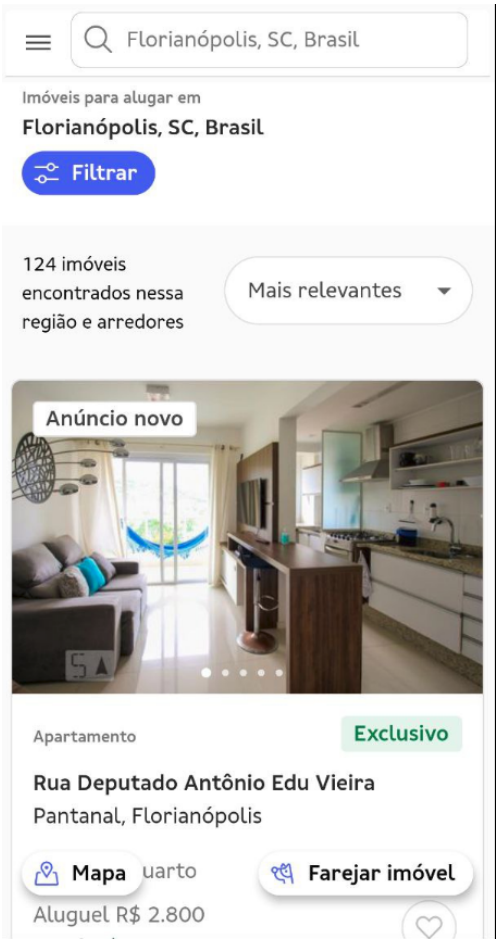
A empresa disponibiliza seu serviço através do website https://www.quintoandar.com.br/ e do aplicativo móvel QuintoAndar disponível na Play Store do Google e na App Store da Apple, conforme mostra as Figuras 7 e 8.

Figura 7 - Captura de tela da lista de imóveis do website Quinto Andar



Fonte: https://www.quintoandar.com.br/ (2021)

Figura 8 - Captura de tela da lista de imóveis do aplicativo móvel Quinto Andar



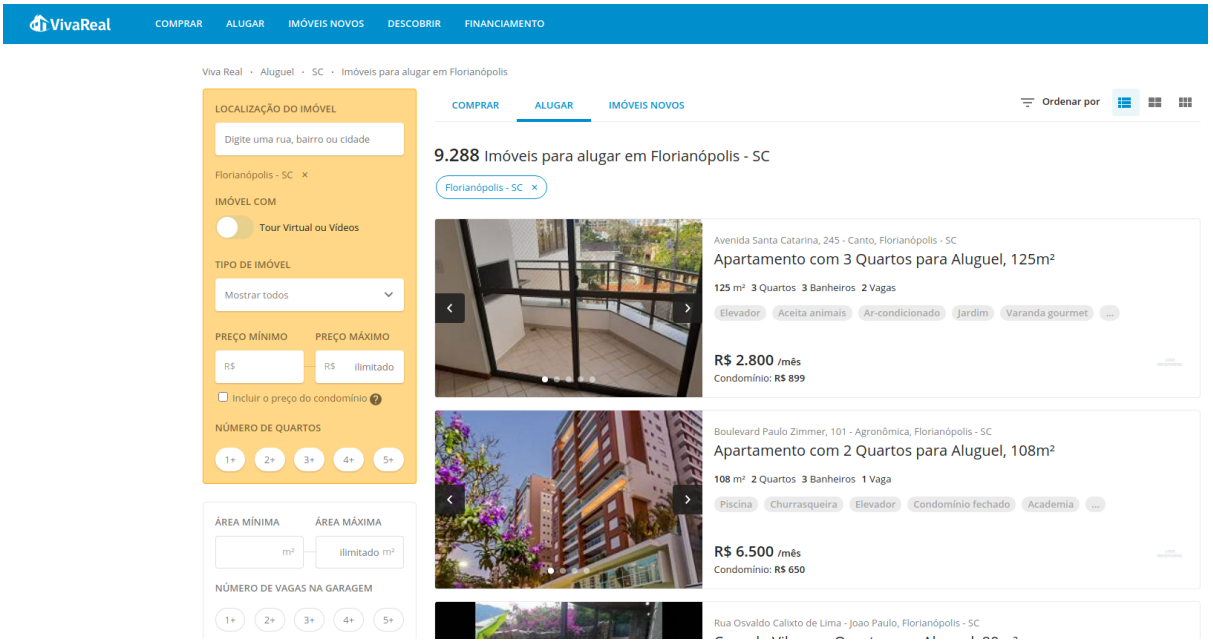
Fonte: https://www.quintoandar.com.br/ (2021)

O serviço mostra os anúncios em formato de lista ou em formato de mapa, conforme escolhido pelo usuário. Também fornece uma variedade de filtros que ajudam o usuário a escolher os melhores imóveis, como por exemplo: proximidade do metrô, se é mobiliado ou não, qual tipo de mobília, instalação e itens que existem no imóvel.

3.5 VivaReal

VivaReal é uma empresa do Grupo ZAP de anúncios de imóveis, contando com anúncios para aluguel e venda de imóveis. O serviço é disponibilizado no website https://www.vivareal.com.br/ e também na Play Store do Google e App Store da Apple na forma de aplicativo móvel.

Figura 9 - Captura de tela da lista de imóveis do website VivaReal

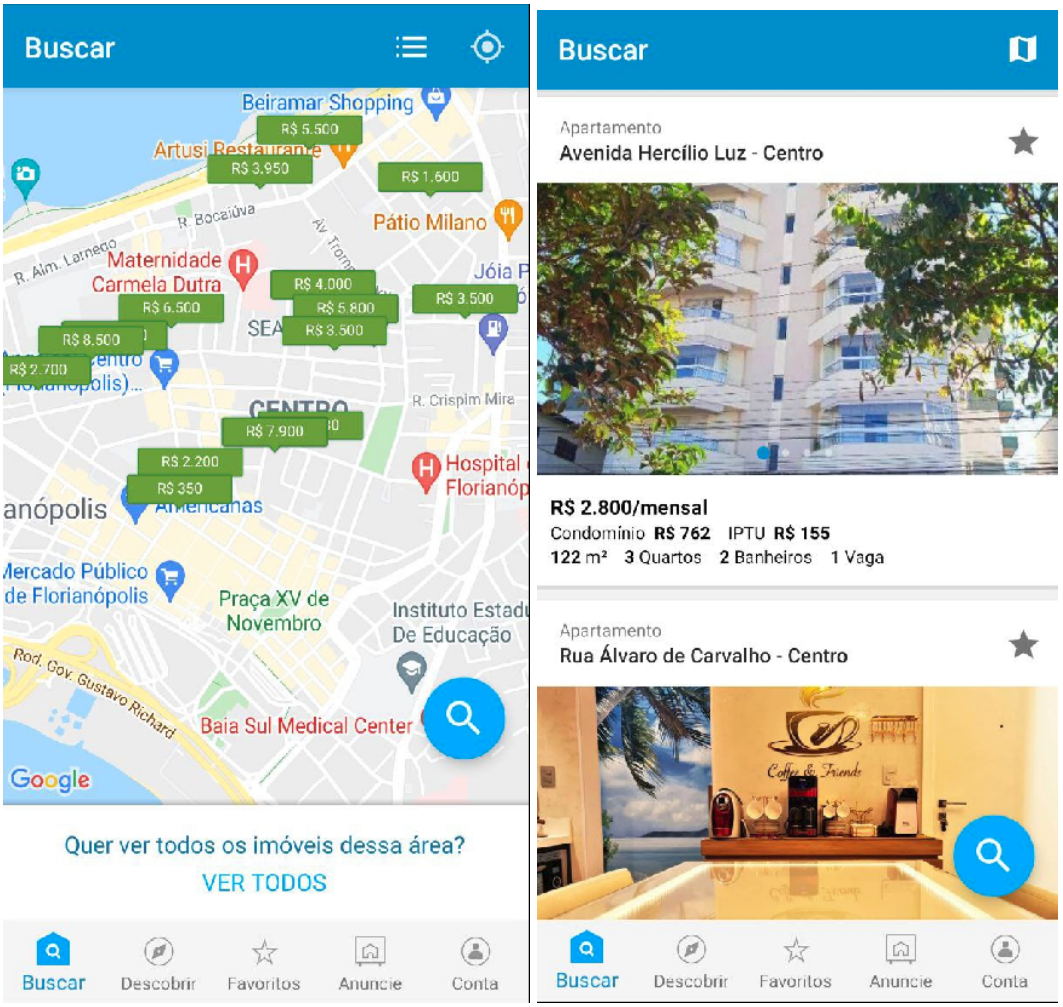


Fonte: https://www.vivareal.com.br/ (2021)

No website, conforme mostra a Figura 9, os anúncios são mostrados em formato de lista com múltiplos filtros disponíveis como por exemplo: comodidades e serviços, segurança, número de banheiros, entre outros. Um filtro que se destaca é a opção de incluir o preço do condomínio no filtro de preço.

Na versão móvel do serviço, conforme mostra a Figura 10, além da listagem em formato de lista, há também um mapa com a localização dos imóveis disponíveis, conforme a imagem abaixo.

Figura 10 - Captura de tela da lista de imóveis do aplicativo móvel VivaReal



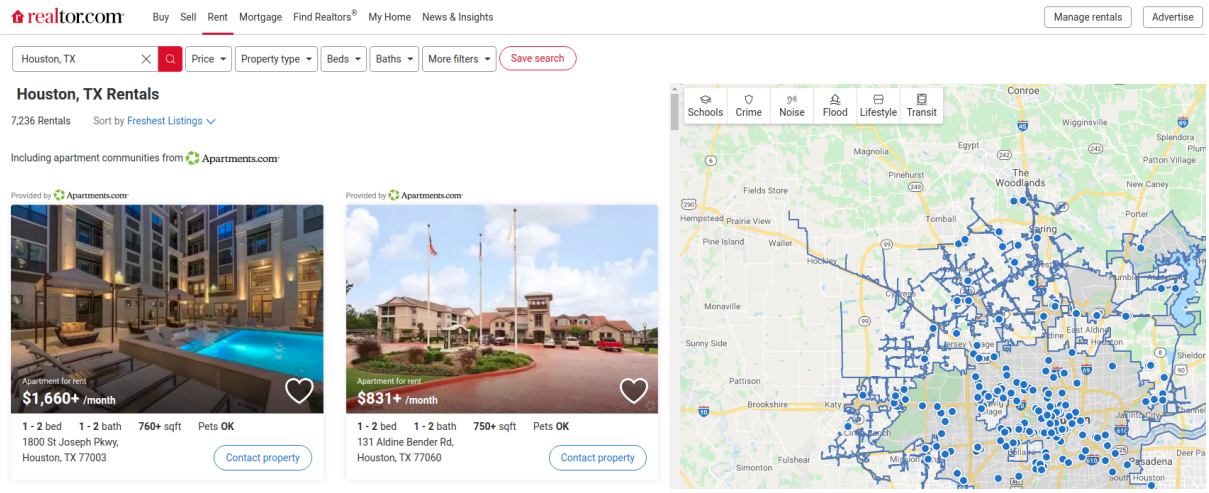
Fonte: https://www.vivareal.com.br/ (2021)

3.7 Realtor

Realtor é um dos maiores serviços de anúncios de aluguel e venda de imóveis dos Estados Unidos, disponível pelo website https://www.realtor.com/. O serviço é oferecido também através de um aplicativo móvel disponível na Play Store do Google e App Store da Apple, conforme mostra as Figuras 11 e 12.

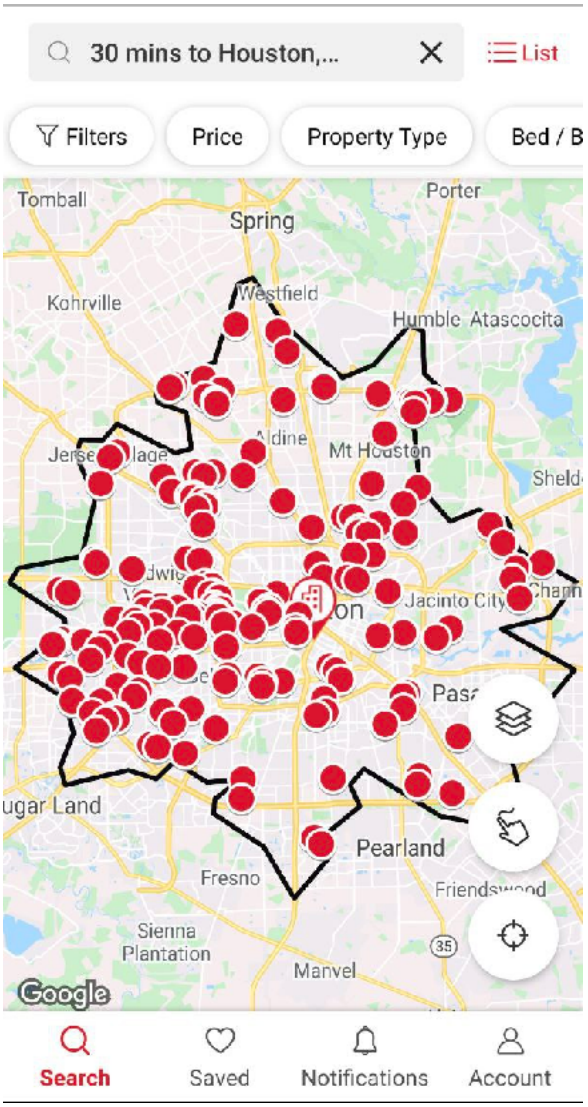
O serviço fornece os mesmos filtros mencionados nos itens anteriores, mas na sua versão móvel há uma opção para o usuário filtrar os anúncios por uma distância máxima, em minutos, entre o imóvel e seu trabalho, assim mostrando apenas os anúncios dentro deste tempo.

Figura 11 - Captura de tela da lista de imóveis do website Realtor



Fonte: https://www.realtor.com/ (2021)

Figura 12 - Captura de tela do mapa de imóveis do aplicativo móvel Realtor



Fonte: https://www.realtor.com/ (2021)

3.8 Comparativo

Percebe-se que nos serviços voltados para universitários como Classificados UFSC, Federal SC e Morar USP faltam muitos filtros úteis para os imóveis comparado com Quinto Andar, VivaReal e Realtor. Fica clara a diferença tecnológica entre as soluções voltadas para universitários e as voltadas para o mercado geral.

O único serviço que provê algum filtro de distância é o Realtor. Não foi possível encontrar nenhum filtro parecido nos serviços brasileiros pesquisados. Quase todos os serviços têm um ponto positivo, mas não há uma junção de todos estes pontos em um só aplicativo, ainda mais para o público universitário, conforme a tabela abaixo retrata.

Tabela 1 - Comparativo entre os serviços

Serviço	Público	Filtros avançados	Possui Mapa	Filtro por distância	Possui App
Classificados UFSC	Universitários	Não	Não	Não	Não
Federal SC	Universitários	Não	Não	Não	Não
Morar USP	Universitários	Não	Sim	Não	Não
QuintoAndar	Geral	Sim	Sim	Não	Sim
VivaReal	Geral	Sim	Sim	Não	Sim
Realtor	Geral	Sim	Sim	Sim	Sim

Fonte: O autor (2021)

4. PROPOSTA DA SOLUÇÃO

Este projeto propõe criar um aplicativo móvel de anúncio de imóveis focado para o público universitário. Primeiramente o usuário escolherá em qual centro ele estuda na universidade, por exemplo: Centro Tecnológico da Universidade Federal de Santa Catarina. Com isso, ele irá visualizar os imóveis próximos da região do centro, tanto no formato de mapa, onde os imóveis aparecerão como pontos no mapa, quanto no formato de lista. Os anúncios poderão ser filtrados utilizando-se os filtros já conhecidos de outras ferramentas. Além disso, será possível filtrar pela distância em minutos a pé, de bicicleta, carro e transporte público entre o imóvel e o centro de estudo. Ao clicar em um anúncio o usuário poderá ver as informações necessárias, como descrição, fotos, características e contato.

Para que existam anúncios no aplicativo, também serão desenvolvidas funcionalidades para o locatário, onde este usuário poderá cadastrar e gerenciar seus anúncios.

4.1 Idealização

O projeto consiste em duas grandes partes, um cliente e um servidor. O servidor será um serviço remoto responsável por armazenar informações sobre os usuários e seus anúncios. A principal responsabilidade deste servidor será receber as requisições de cadastro de anúncios e calcular a distância entre o imóvel da requisição e os centros cadastrados da universidade, e armazenar o anúncio junto do cálculo. Além disso, ele irá prover algumas outras funcionalidades necessárias para criação, leitura e edição dos anúncios. Essas requisições serão feitas através de uma interface REST API.

A parte cliente será capaz de consumir os dados através da interface mencionada, lendo e enviando os dados necessários. Será responsabilidade do cliente mostrar tais dados para o usuário através de uma interface gráfica de forma organizada, também será responsabilidade do cliente prover filtros para que os dados sejam filtrados na interface gráfica.

4.2 Casos de Uso

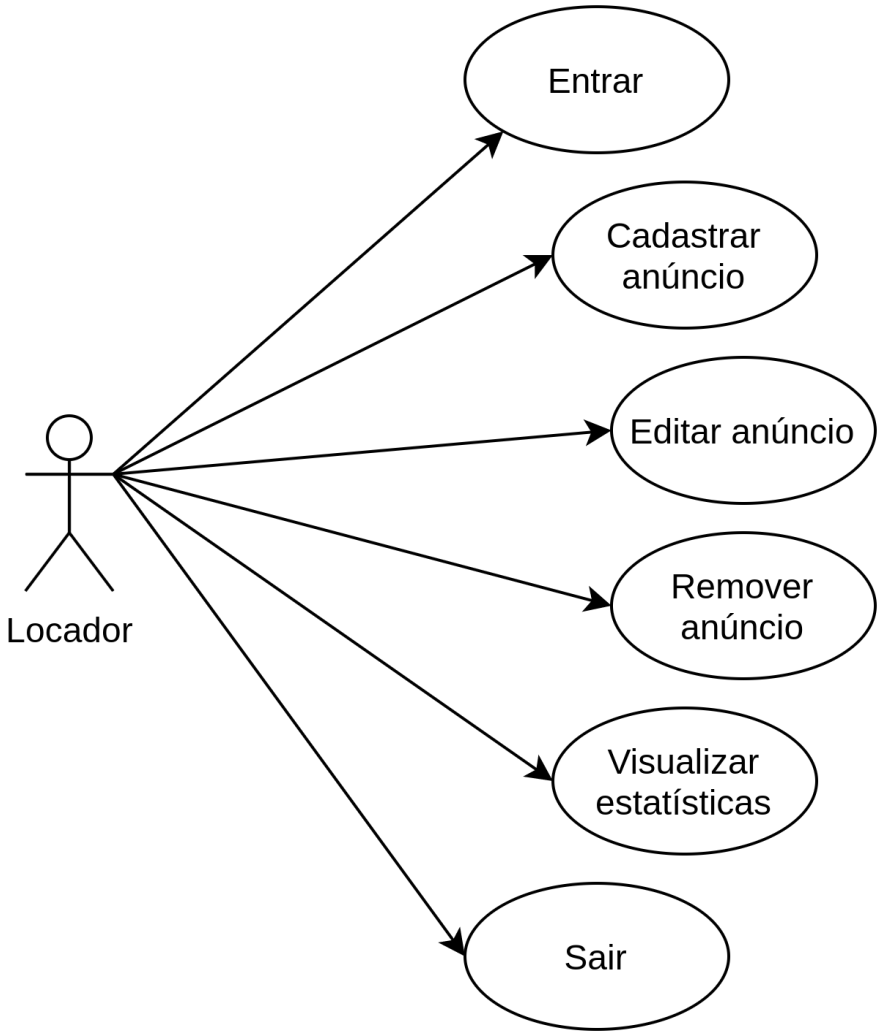
Casos de uso são uma lista de ações ou eventos que normalmente definem as interações entre atores de um sistema para atingir um objetivo. Foram

identificados 2 atores, o locatário, que é a pessoa que está procurando um imóvel, e o locador, que é a pessoa que quer anunciar seu imóvel. Os casos de uso foram identificados pelo autor apenas, os usuários não foram consultados nesta etapa.

4.2.1 Locador

Para o locador foram identificados 6 casos de uso, conforme a Figura 13.

Figura 13 - Casos de uso para o locador



Fonte: O autor (2021)

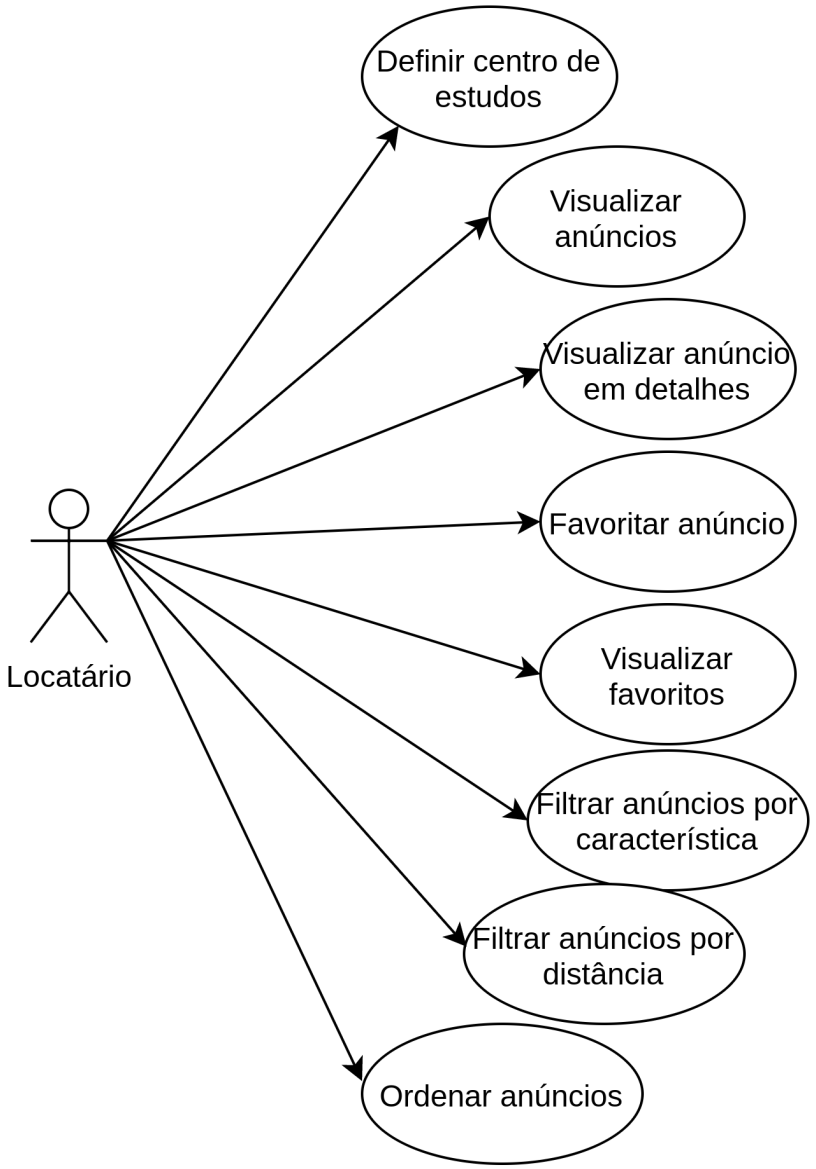
1. **Entrar:** Capacidade de um locador entrar no aplicativo com uma conta já existente.

2. **Cadastrar anúncio:** Capacidade de um locador cadastrar um anúncio, com características, localização e fotos do imóvel. Apenas disponível se o usuário estiver *logado* no aplicativo.
3. **Editar anúncio:** Capacidade de um locador editar um anúncio previamente cadastrado por ele mesmo. Apenas disponível se o usuário estiver *logado* no aplicativo.
4. **Remover anúncio:** Capacidade de um locador remover um anúncio previamente cadastrado por ele mesmo. Apenas disponível se o usuário estiver *logado* no aplicativo.
5. **Visualizar estatísticas:** Capacidade de um locador visualizar estatísticas do perfil de locadores interessados nos seus anúncios cadastrados. Apenas disponível se o usuário estiver *logado* no aplicativo.
6. **Sair:** Capacidade de um locador sair de sua conta.

4.2.2 Locatário

Para o locatário foram identificados, 8 casos de uso, conforme a Figura 14.

Figura 14 - Casos de uso para o locatário



Fonte: O autor (2021)

1. **Definir centro de estudos:** Capacidade de um locatário definir o seu centro da universidade, por exemplo: Centro Tecnológico.
2. **Visualizar anúncios:** Capacidade de um locatário visualizar os anúncios disponíveis, em formato de lista ou mapa.
3. **Visualizar anúncio em detalhes:** Capacidade de um locatário visualizar um anúncio em detalhes, podendo ver sua descrição e características.
4. **Favoritar anúncio:** Capacidade de um locatário adicionar um anúncio a sua lista de anúncios favoritos, para serem visualizados novamente em outro

momento. Esta lista é armazenada na memória local do aplicativo.

5. **Visualizar favoritos:** Capacidade de um locatário visualizar sua lista de anúncios favoritos em formato de lista. Esta lista é armazenada na memória local do aplicativo.
6. **Filtrar anúncios por características:** Capacidade de um locatário filtrar a lista ou mapa de anúncios através das características dos anúncios, por exemplo: faixa de preço, quantidade de quartos.
7. **Filtrar anúncios por distância (minutos):** Capacidade de um locatário filtrar a lista ou mapa de anúncios através da distância em minutos a pé ou de bicicleta entre o centro de estudos e imóvel.
8. **Ordenar anúncios:** Capacidade de um locatário ordenar a lista de anúncios pelo preço ou distância em minutos.

4.3 Wireframes

Um *wireframe*, também conhecido como projeto de tela, é um guia visual que representa a estrutura do esqueleto de uma interface gráfica (BALSAMIQ). *Wireframes* são criados com a finalidade de organizar os elementos para melhor cumprir um determinado propósito. O *wireframe* descreve o layout da página. O *wireframe* geralmente não tem estilo tipográfico, cor ou gráficos, já que o foco principal está na funcionalidade, comportamento e prioridade do conteúdo.

A seguir serão apresentados os *wireframes* do projeto, os quais foram divididos em duas partes, locador e locatário, pois são atores diferentes. A única tela em comum é a tela inicial do aplicativo, conforme ilustra a Figura 15. Este *wireframe* é relacionado ao caso de uso “Definir centro de estudos”. Os *wireframes* desenvolvidos não foram validados por usuários, foram apenas validados pelos autores.

Figura 15 - *Wireframe* da tela inicial

Fonte: O autor (2021)

A tela inicial tem como objetivo dar boas vindas ao usuário e fazer um processo de *onboarding*, que nada mais é que apresentar as funcionalidades e fazer as configurações iniciais. Neste caso o usuário deve selecionar seu centro de estudos, e clicar no botão para ir para a próxima tela, que é tratada na seção 4.3.2. Caso o usuário seja um locador, ele poderá clicar no botão ao fim da tela para ir para as configurações de um usuário locador. Essas telas são tratadas na seção 4.3.1.

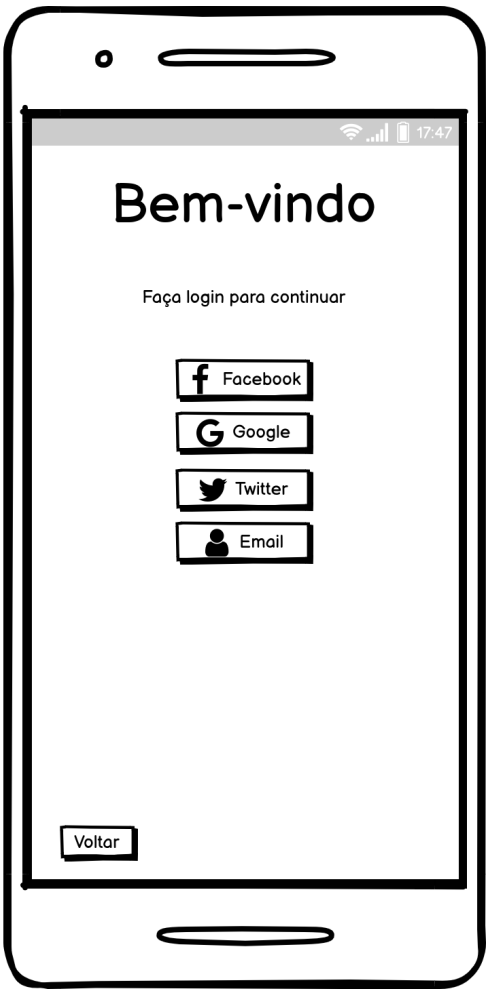
4.3.1 Locador

A tela proposta pela Figura 16 contém a funcionalidade de login, representada pelo caso de uso “Entrar”, pois para que o locador consiga cadastrar anúncios na plataforma ele precisa de uma conta para que os anúncios sejam atrelados a um usuário.

Para evitar a necessidade de implementar uma funcionalidade de cadastro, foi optado por fazer login através de redes sociais, utilizando ferramentas já existentes para isso.

Caso o usuário deseje voltar para a tela inicial, ele poderá clicar no botão de voltar ao fim da tela.

Figura 16 - *Wireframe* da tela de login



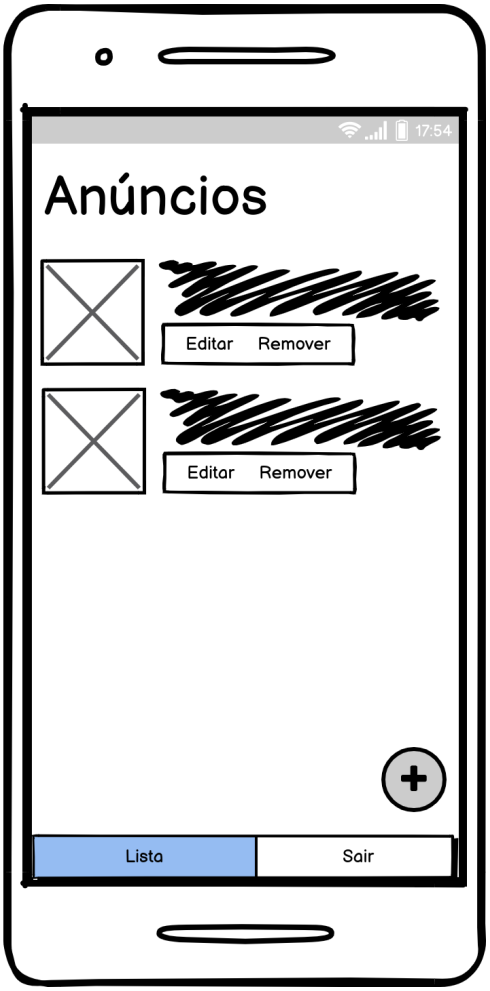
Fonte: O autor (2021)

Ao efetuar login, o usuário será direcionado para a próxima tela, que é ilustrada pela Figura 17. Nesta tela o usuário poderá visualizar os anúncios já cadastrados, podendo também editá-los ou removê-los. Este *wireframe* é relacionado aos casos de uso “Cadastrar anúncio”, “Editar anúncio” e “Remover anúncio”.

Caso o usuário queira inserir um novo anúncio, ele poderá clicar no botão de adicionar no canto inferior direito da tela, que o levará para a tela de cadastro de anúncio. Esta mesma tela de cadastro de anúncio é mostrada para o usuário caso ele deseje editar um anúncio.

Nesta tela também há a opção de sair de sua conta, clicando no botão de sair no canto inferior direito da tela.

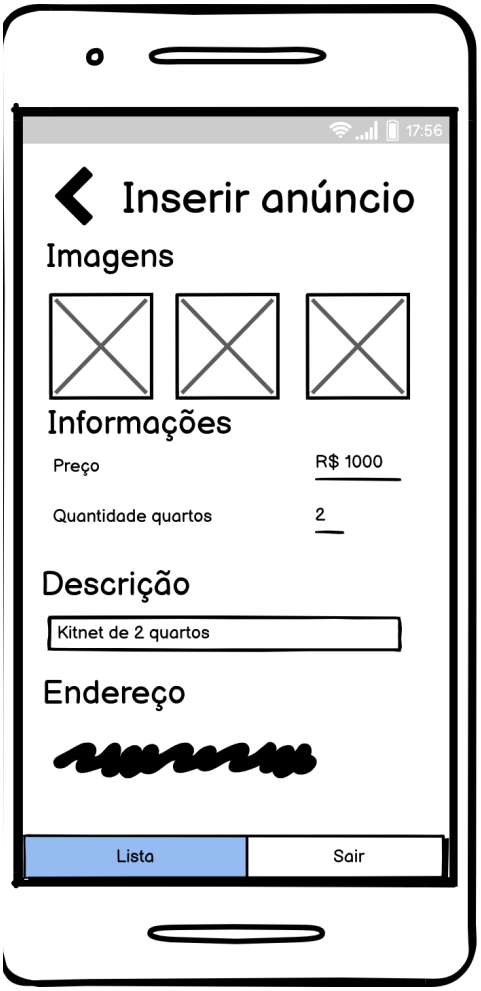
Figura 17 - *Wireframe* da lista de anúncios



Fonte: O autor (2021)

A Figura 18 representa a tela de cadastro mencionada anteriormente. Nesta tela o usuário poderá inserir imagens do imóvel, adicionar informações, cadastrar o endereço, e por fim uma descrição do imóvel. Este *wireframe* é relacionado aos casos de uso “Cadastrar anúncio” e “Editar anúncio”.

Figura 18 - *Wireframe* da criação/edição de anúncio



Fonte: O autor (2021)

4.3.2 Locatário

Na tela representada pela Figura 19, o usuário verá os imóveis disponíveis no mapa através de marcadores, que quando clicados levam para a tela de detalhes do imóvel, como mostra a Figura 20. Este *wireframe* é relacionado aos casos de uso “Visualizar anúncios”, "Filtrar anúncios por distância” e “Filtrar anúncios por caracteristica”.

Além disso, há um menu superior onde ele poderá aplicar filtros aos anúncios, onde aparecerão apenas os marcadores de imóveis que se encaixarem nos filtros selecionados. Também há um menu inferior, que navega para outras telas; o botão Lista, que leva para a tela de anúncios em formato de lista, mostrada na Figura 20; e o botão Favoritos, que leva para a tela de anúncios favoritos, como mostra a Figura 21.

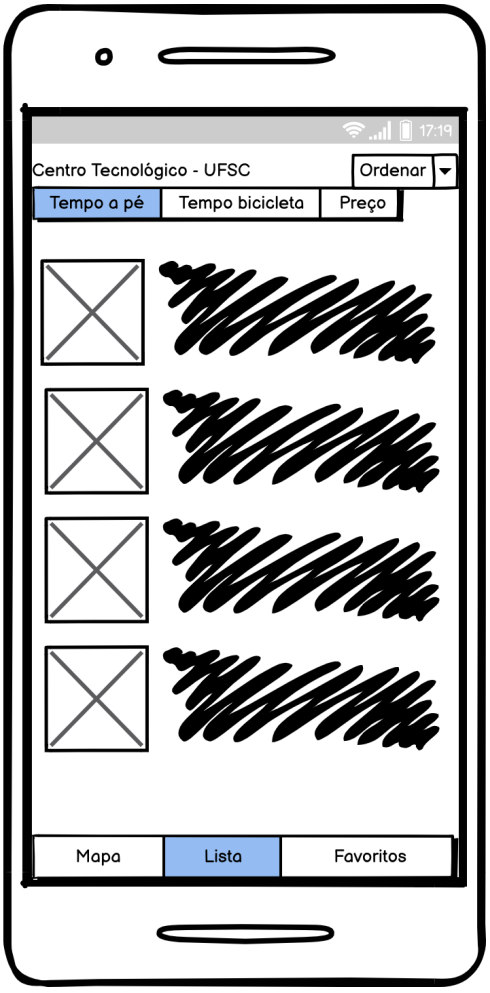
Figura 19 - *Wireframe* da listagem de imóveis no mapa

Fonte: O autor (2021)

A Figura 20 apresenta a tela de anúncios em formato de lista, conforme mencionado anteriormente. Nesta tela o usuário poderá ver uma prévia de cada anúncio, como imagens e informações básicas. Este *wireframe* é relacionado aos casos de uso “Visualizar anúncios”, "Filtrar anúncios por distância” e “Filtrar anúncios por caracteristica”.

Os mesmos filtros disponíveis na tela do Mapa estão disponíveis aqui. Por ser uma listagem, há a opção de ordená-la, que está disponível no canto superior direito da tela. Ao clicar em um anúncio, o usuário será direcionado para a tela de detalhes do anúncio, representada pela Figura 22.

Figura 20 - *Wireframe* da listagem de imóveis

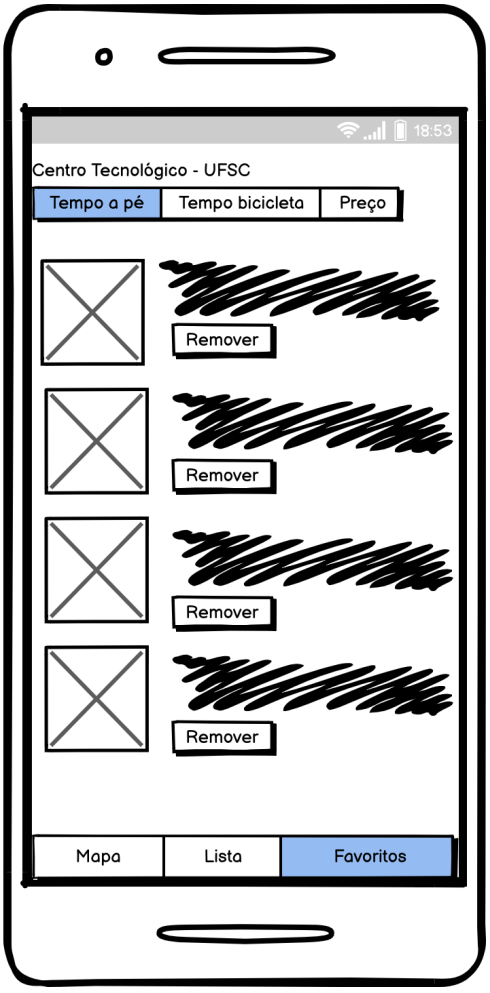


Fonte: O autor (2021)

A Figura 21 apresenta a tela de anúncios favoritos do usuário. Estes anúncios ficam salvos localmente no aplicativo e não na conta do usuário. Os filtros mencionados anteriormente também ficam disponíveis nesta tela. Este *wireframe* é relacionado aos casos de uso “Visualizar favoritos”, "Filtrar anúncios por distância” e “Filtrar anúncios por caracteristica”.

Ao clicar em um anúncio, o usuário será direcionado para a tela de detalhes do anúncio, representada pela Figura 22. Caso o usuário queira remover um anúncio de sua lista de favoritos, ele poderá clicar no botão Remover, que está disponível em cada item da lista.

Figura 21 - *Wireframe* da listagem de favoritos

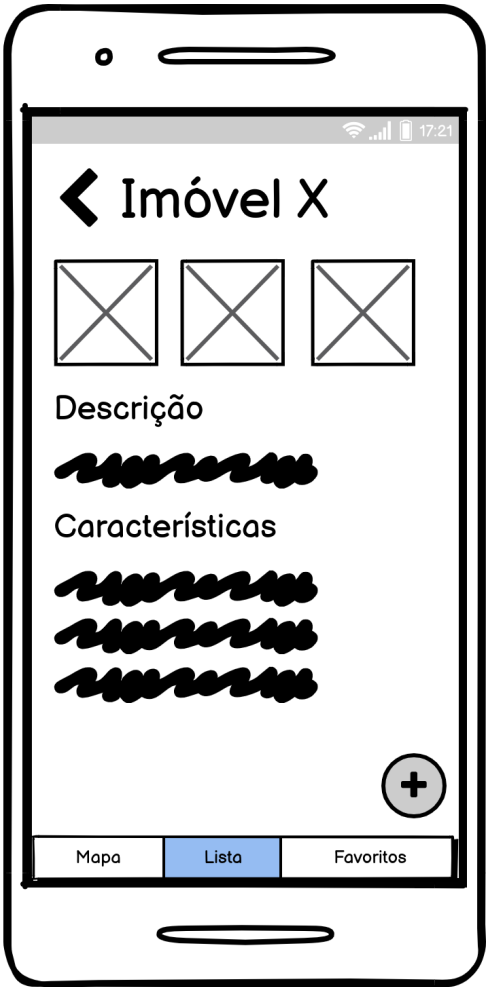


Fonte: O autor (2021)

A tela de detalhes do anúncio é representada pela Figura 22. Nesta tela o usuário poderá ver em detalhes o imóvel selecionado, podendo ver múltiplas fotos e várias informações e características do imóvel. Este *wireframe* é relacionado aos casos de uso “Visualizar anúncio em detalhes” e "Favoritas anúncio”.

Caso o usuário goste do anúncio, ele pode clicar no botão de Adicionar, localizado no canto inferior direito da tela. Este botão vai adicionar o anúncio na página de Favoritos do usuário.

Figura 22 - *Wireframe* do imóvel em detalhes



Fonte: O autor (2021)

5. DESENVOLVIMENTO DO SISTEMA

Este capítulo tem como finalidade descrever o processo realizado para o desenvolvimento do sistema.

Primeiramente é descrita a arquitetura do sistema, como os anúncios que serão mostrados no aplicativo foram obtidos, quais tecnologias são utilizadas, e onde elas se encaixam no sistema.

Por fim é apresentado a interface do sistema, onde são mostradas múltiplas figuras com todas as telas do aplicativo juntamente com uma breve descrição de suas funcionalidades.

5.1 Arquitetura

5.1.1 Dados iniciais

Para facilitar o desenvolvimento e testes do aplicativo e ter uma boa quantidade de dados, no caso, anúncios de imóveis, seria ideal ter um conjunto exemplo de anúncios reais. Para isso foi desenvolvida uma ferramenta de *scrapping* em *NodeJS*, que varre os anúncios dos sites *Viva Real* e *Zap Imóveis* e obtém os dados dos anúncios.

Para realizar o *scrapping*, foram inspecionados o código-fonte e as requisições de rede de alguns sites mencionados anteriormente em Estado da Arte. Como resultado dessa inspeção, descobriu-se que os sites *Viva Real* e *Zap Imóveis* utilizam de uma API idêntica para se comunicar com seus servidores, provavelmente por pertencerem ao mesmo grupo de empresas. Logo esses dois sites foram escolhidos pela facilidade de fazer um *scrapping* de uma API ao invés de um HTML.

A ferramenta consulta ambas as APIs, e consolida os anúncios em uma lista única, removendo anúncios duplicados. Os anúncios duplicados são aqueles que têm um mesmo título e descrição. Após isso, é feito o *download* das imagens dos anúncios localmente, e referenciado o caminho dessas imagens no anúncio.

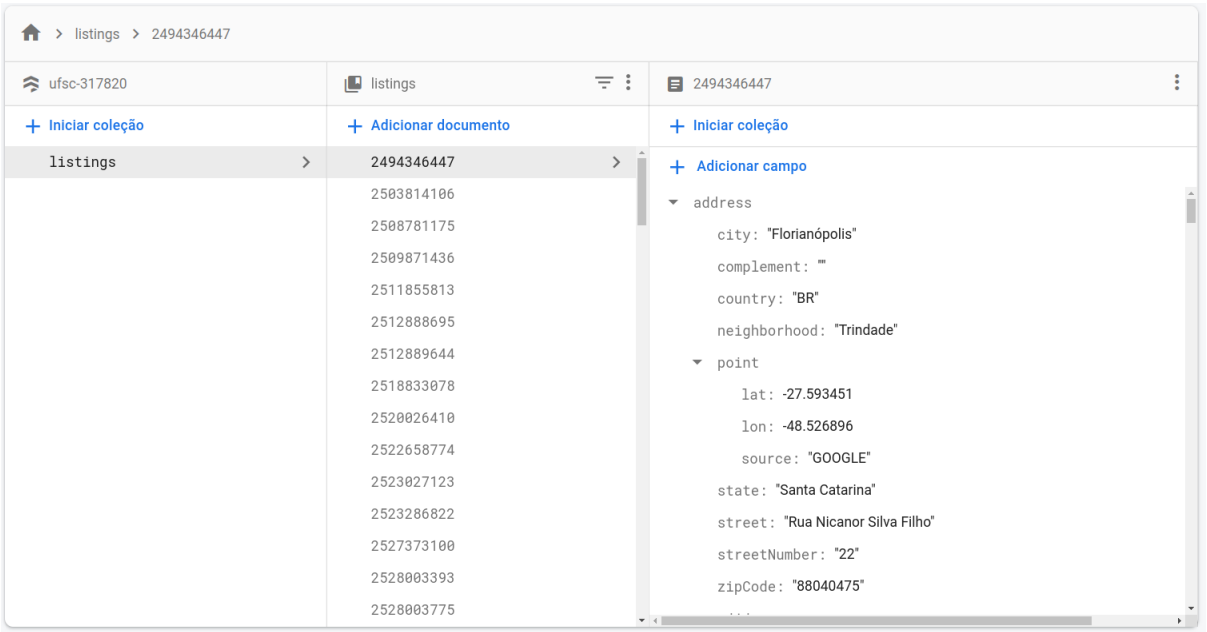
Por fim, a ferramenta envia os anúncios para o banco de dados, e as imagens para o *Object Storage*.

5.1.2 Backend

Para o backend, foram utilizados alguns serviços do *Firebase*. O *Firebase* é uma plataforma desenvolvida pelo *Google* para criar aplicativos móveis e web de forma fácil e rápida. A plataforma foi escolhida pois a sua utilização reduz a quantidade de código necessária para desenvolver o aplicativo, e reduz a complexidade de implementar certas funcionalidades "do zero", como a autenticação, por exemplo.

Primeiramente, foi utilizado o serviço *Cloud Firestore*, que é um banco de dados NoSQL flexível e escalável para armazenar e sincronizar dados para desenvolvimento do lado do cliente e do servidor. A escolha foi dada pela praticidade de uso e integração com os outros serviços do *Firebase* e não por ser um banco de dados NoSQL especificamente. A figura 23 representa um objeto, um anúncio, na coleção de anúncios.

Figura 23 - Coleção de anúncios no *Cloud Firestore*

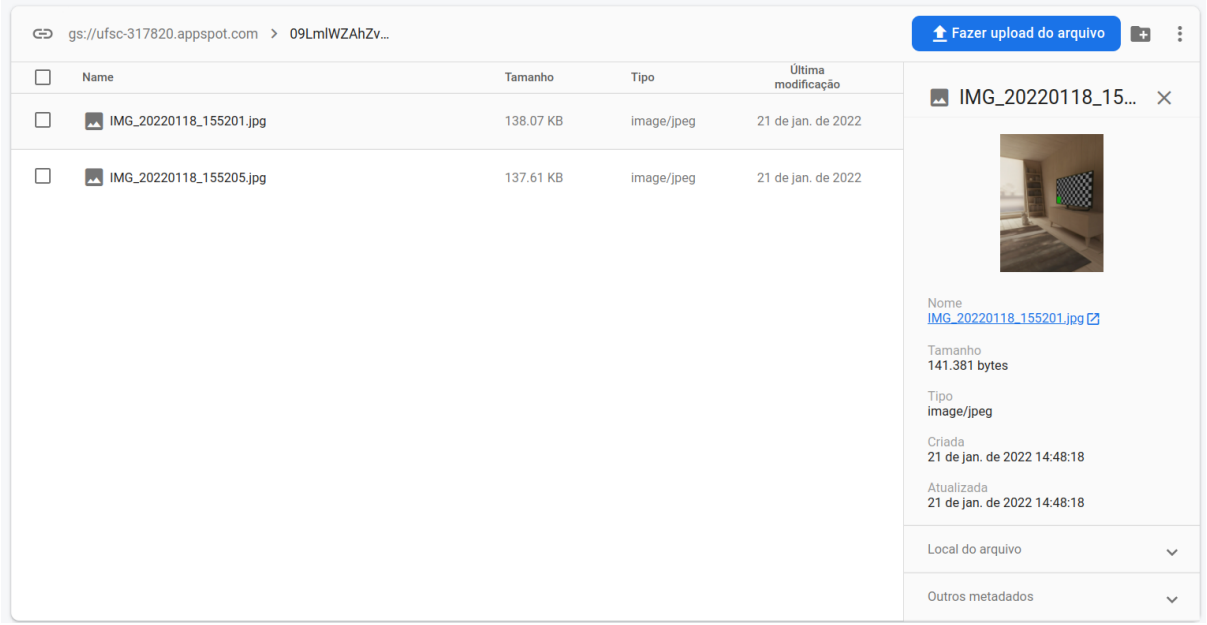


Fonte: O autor (2022)

Como é necessário armazenar as fotos dos anúncios, foi utilizado o *Cloud Storage* para isso, o qual é um *Object Storage*, que armazena os arquivos em formato de objeto ao invés de armazenar no sistema de arquivos. Ele também gera

um *link* para cada objeto, que pode ser usado pelo aplicativo para exibir a imagem. A figura 24 representa uma pasta no *Object Storage*.

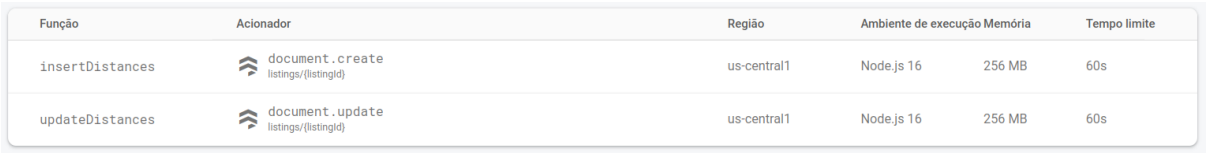
Figura 24 - Pasta de imagens no *Cloud Storage*



Fonte: O autor (2022)

Para tratar das requisições, foi utilizado o *Cloud Functions*, o qual é uma estrutura *serverless* que permite executar automaticamente o código de *backend* em resposta a eventos acionados por recursos do *Firebase* e solicitações HTTPS. A figura 25 mostra as duas *functions* cadastradas, as quais são descritas em seguida.

Figura 25 - *functions* cadastradas no *Cloud Functions*

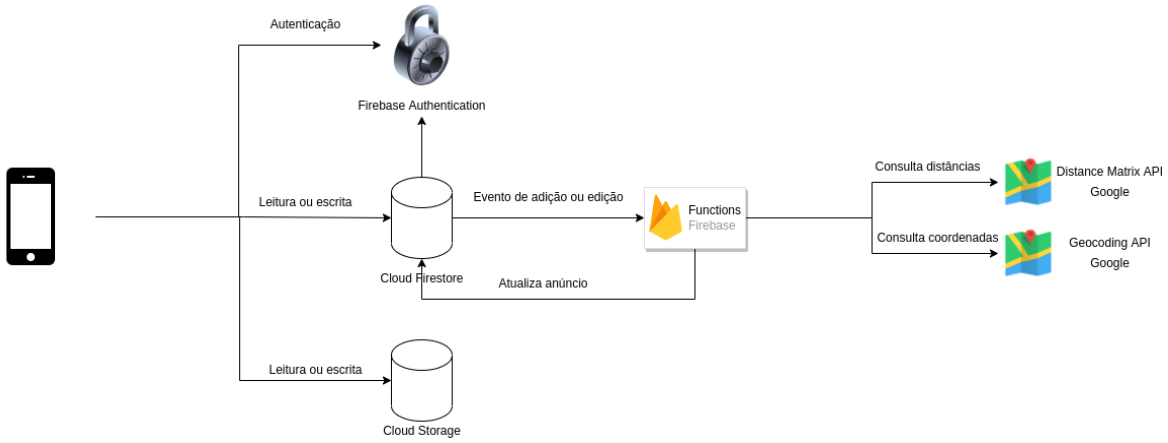


Fonte: O autor (2022)

Para autenticação e autorização, foi utilizado o *Firebase Authentication*, o qual oferece serviços de *backend*, *SDKs* fáceis de usar e bibliotecas de *UI* prontas para autenticar usuários no aplicativo. Ele suporta autenticação usando senhas, números de telefone, provedores de identidade federados populares como Google, Facebook e Twitter.

Com o uso destes serviços a quantidade de código para o backend foi reduzida consideravelmente. A figura 26 representa o fluxo de dados no *backend.*

Figura 26 - Fluxo de dados no *backend*



Fonte: O autor (2022)

O aplicativo se comunica apenas com o *Cloud Firestore* e com *Firebase Authentication*. Quando o usuário é um locatário, o aplicativo conversa diretamente com o *Cloud Firestore*, utilizando o SDK fornecido, sem que haja necessidade de criação de uma API. Em seguida, o aplicativo busca todos os anúncios cadastrados e os mostra na interface do aplicativo.

Para o caso de um cadastro de imóvel por um locador, primeiramente, o aplicativo se comunica com o *Firebase Authentication* para criar um usuário, ou entrar com um usuário já existente. Após feito o login e preenchido o formulário de cadastro de um novo imóvel, o aplicativo se comunica com o *Cloud Firestore* e envia os dados para cadastro no banco de dados.

Assim que o banco de dados recebe esse novo cadastro, o *Cloud Functions* recebe um evento com a informação de que um item foi cadastrado no banco de dados, recebendo também este item. Com essas informações, então, a *function* cadastrada recebe os dados do imóvel, consulta a API do *Google Maps* para definir as coordenadas de latitude e longitude, além de calcular as distâncias entre o imóvel e os centros de estudos. Por fim, com esses novos dados da API, a *function*

atualiza os dados do imóvel no banco de dados, adicionando a posição e as distâncias.

Como o aplicativo acessa diretamente o banco de dados, qualquer usuário poderia inserir ou deletar anúncios. Para resolver este problema, o *Firebase* disponibiliza regras que podem ser adicionadas ao banco de dados. Foram adicionadas as seguintes regras: qualquer usuário pode ler anúncios, apenas usuários cadastrados podem inserir anúncios, e apenas o usuário que criou um anúncio pode alterá-lo ou removê-lo.

5.1.2.1 Distance Matriz API

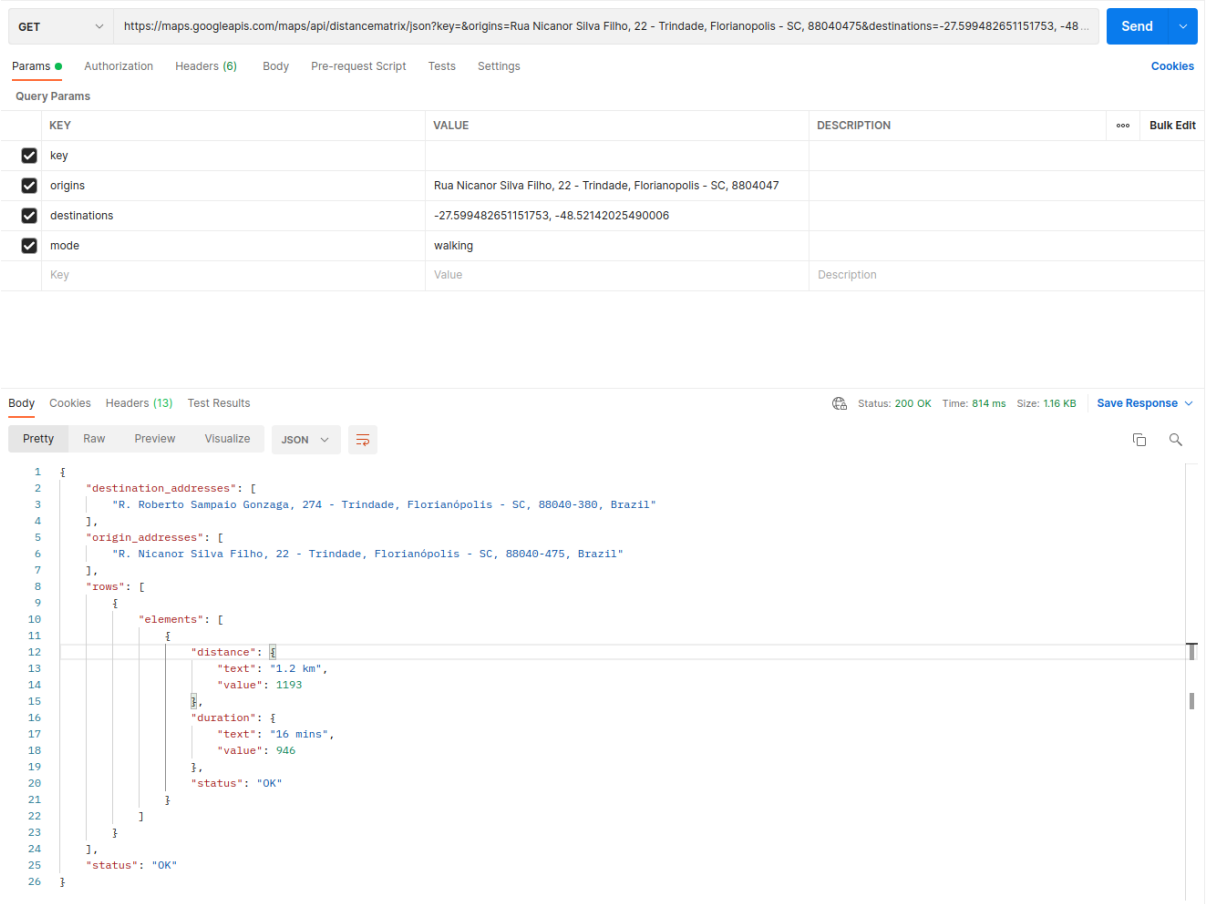
A Distance Matrix API é um serviço que fornece distância e tempo de viagem para uma matriz de origens e destinos em um certo modo de transporte. A API retorna informações com base na rota recomendada entre os pontos inicial e final, conforme calculado pela API do Google Maps.

No caso da *function*, é usado como ponto de origem o endereço do imóvel, e como pontos de destinos são utilizadas as coordenadas de todos os centros da UFSC, assim, ao consultar a API, a resposta contém a distância e tempo entre o imóvel e os vários centros.

Como a API calcula apenas um modo de transporte por vez, para cada anúncio cadastrado são feitas duas chamadas para a API, uma para calcular o tempo dos trajetos usando a bicicleta e outra a pé.

A figura 27 mostra um exemplo de consulta à API.

Figura 27 - Requisição e resposta da API no *Postman*



Fonte: O autor (2022)

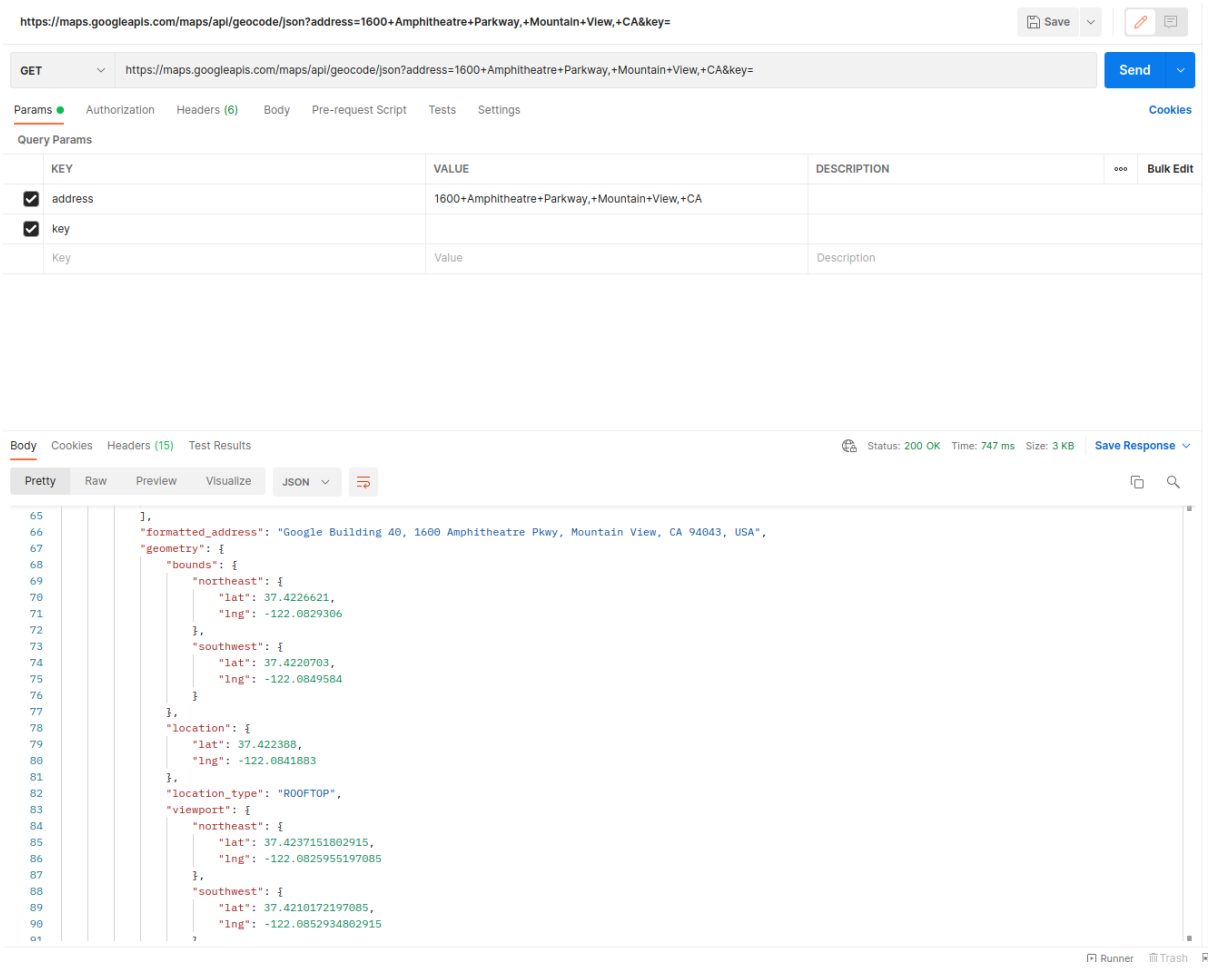
5.1.2.2 Geocoding API

A Geocoding API é um serviço que fornece a geocodificação de endereços. Geocodificação é o processo de conversão de endereços em coordenadas geográficas, que então podem ser usados para colocar marcadores em um mapa.

No caso da *function*, é enviado o endereço do imóvel para a API, a qual retorna uma coordenada geográfica que então vai ser utilizada pelo aplicativo para adicionar um marcador no mapa para este anúncio.

A figura 28 mostra um exemplo de consulta a esta API.

Figura 28 - Requisição e resposta da API no *Postman*



Fonte: O autor (2022)

5.1.3 Aplicativo

O aplicativo foi desenvolvido em *Flutter*, utilizando a biblioteca *provider* para gerenciar os estados do aplicativo. Esta biblioteca facilita a separação de conceitos no desenvolvimento, onde a lógica de negócio é escrita em um *Provider*, fora dos *Widgets*.

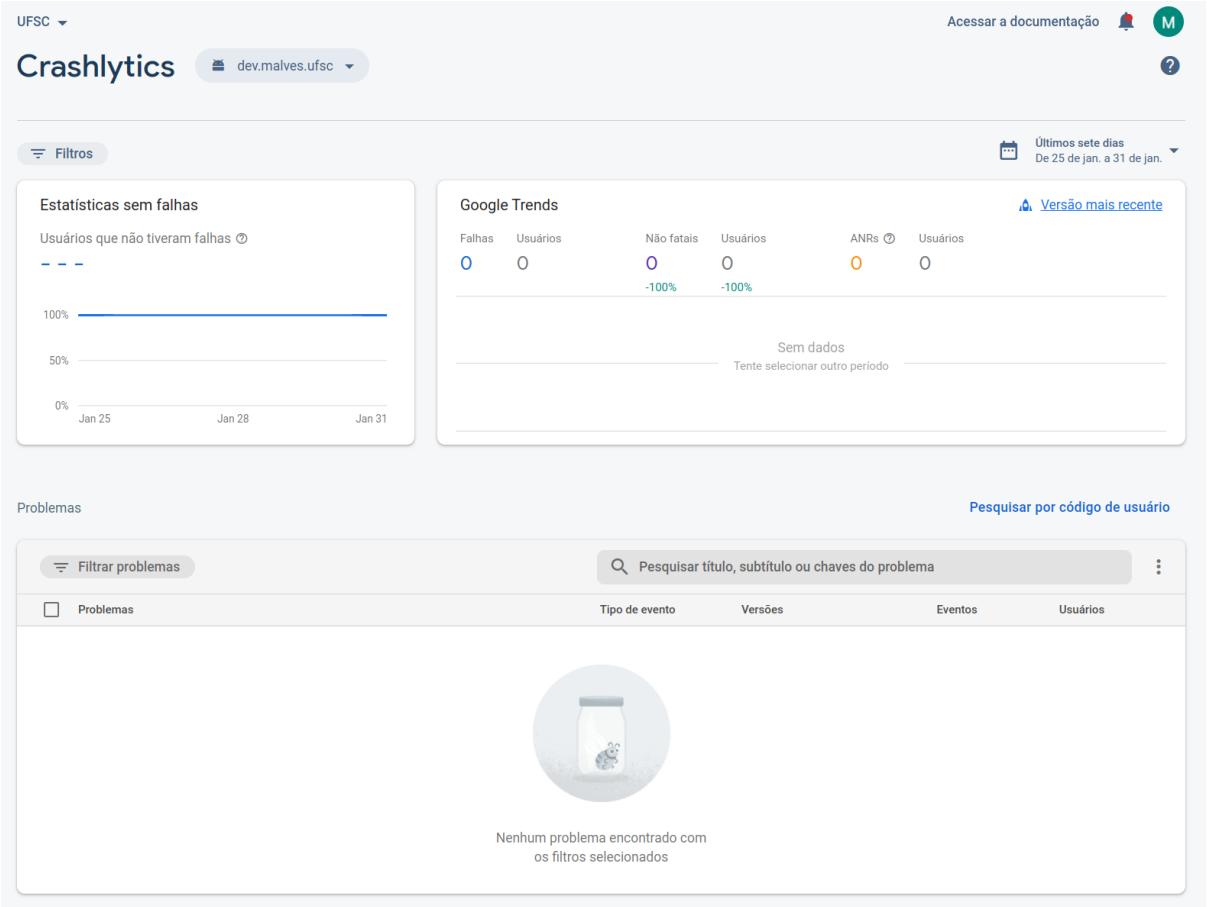
Os *Widgets* então são instanciados abaixo de um certo *Provider* na árvore de renderização, o que significa que um *Provider* pode requisitar que um *Widget* renderize novamente com novos dados. Por exemplo, ao clicar no botão de adicionar anúncio aos favoritos, este evento é enviado do *Widget* para o *Provider*, o qual executa a lógica necessária e agora tem um novo estado, um item a mais na

sua lista de favoritos; com isso, o *Provider* notifica ao *Widget* que ele necessita renderizar novamente considerando os novos dados.

As principais bibliotecas utilizadas foram: *google_maps_flutter*, para desenhar o mapa e adicionar os marcadores; os *SDKs* do *Firebase*, *firebase_core, firebase_auth, cloud_firestore, firebase_storage;* e *google_sign_in*. Estas bibliotecas podem ser visualizadas em https://pub.dev.

Para monitorar *bugs* e travamentos no aplicativo, foi utilizado o *Firebase Crashlytics*, ele é adicionado ao aplicativo em forma de uma biblioteca, a qual inicializa junto ao aplicativo, monitora e gera logs sobre erros que acontecem no aplicativo. Ao encontrar algum erro, este então é enviado para um painel no *Firebase*, onde o desenvolvedor consegue ver a mensagem de erro e mais algumas informações. A figura 29 mostra este painel.

Figura 29 - Painel do *Crashlytics*

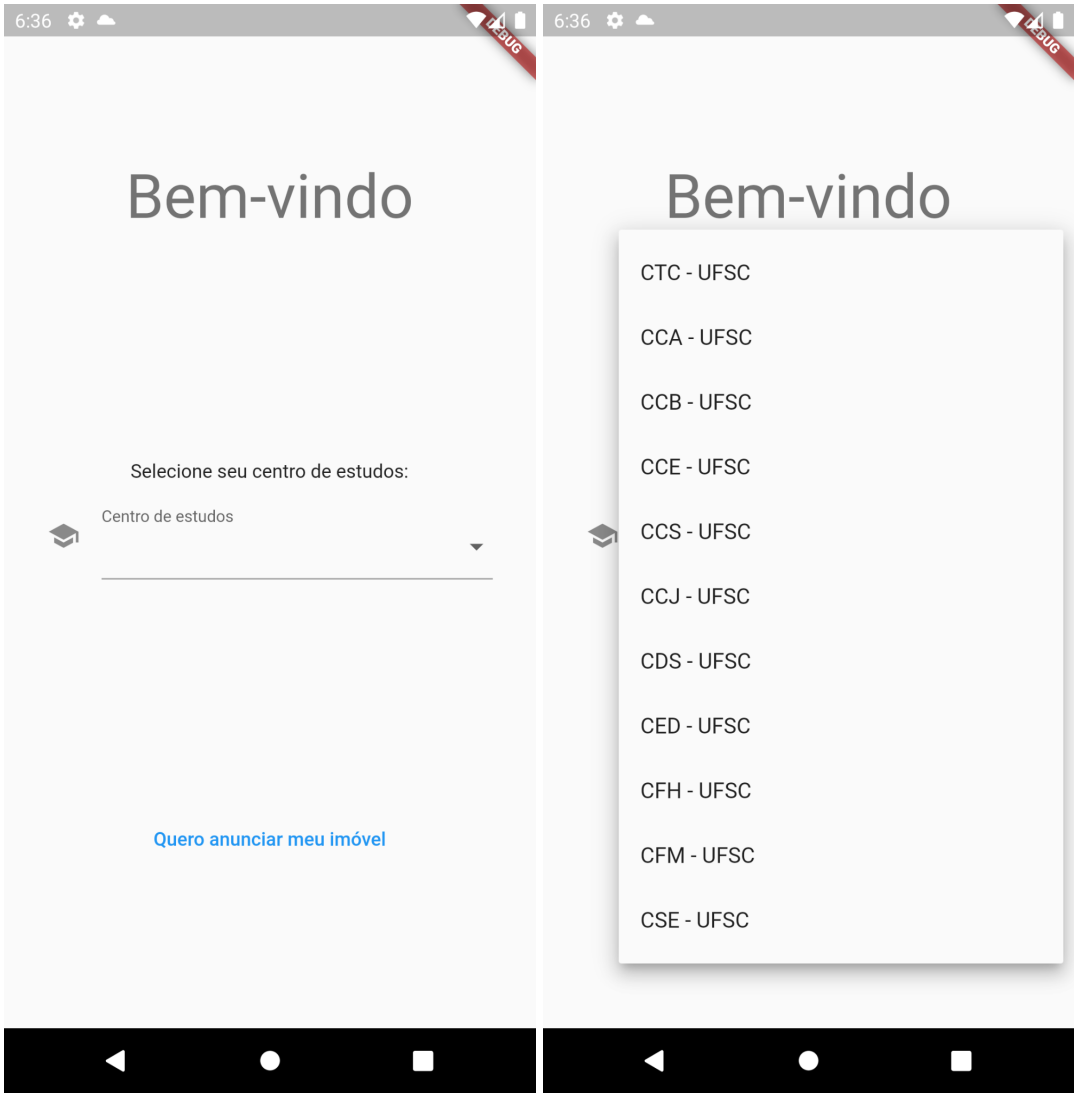


Fonte: O autor (2022)

5.2 Interface do aplicativo

A tela principal do aplicativo, mostrada na figura 30, apresenta duas opções para o usuário: selecionar um centro de estudos para entrar como um locatário, ou pressionar o botão “Quero anunciar meu imóvel” para entrar como um locador.

Figura 30 - Telas iniciais do aplicativo



Fonte: O autor (2022)

5.2.1 Locatário

Após selecionar um centro de estudos, o locatário é apresentado com um mapa com a localização de todos os imóveis disponíveis. Na barra superior, é mostrado o centro de estudos escolhido, os valores dos principais filtros e no canto direito um botão para acessar a página de filtros. A figura 31 representa esta tela.

Figura 31 - Tela de mapa com marcadores

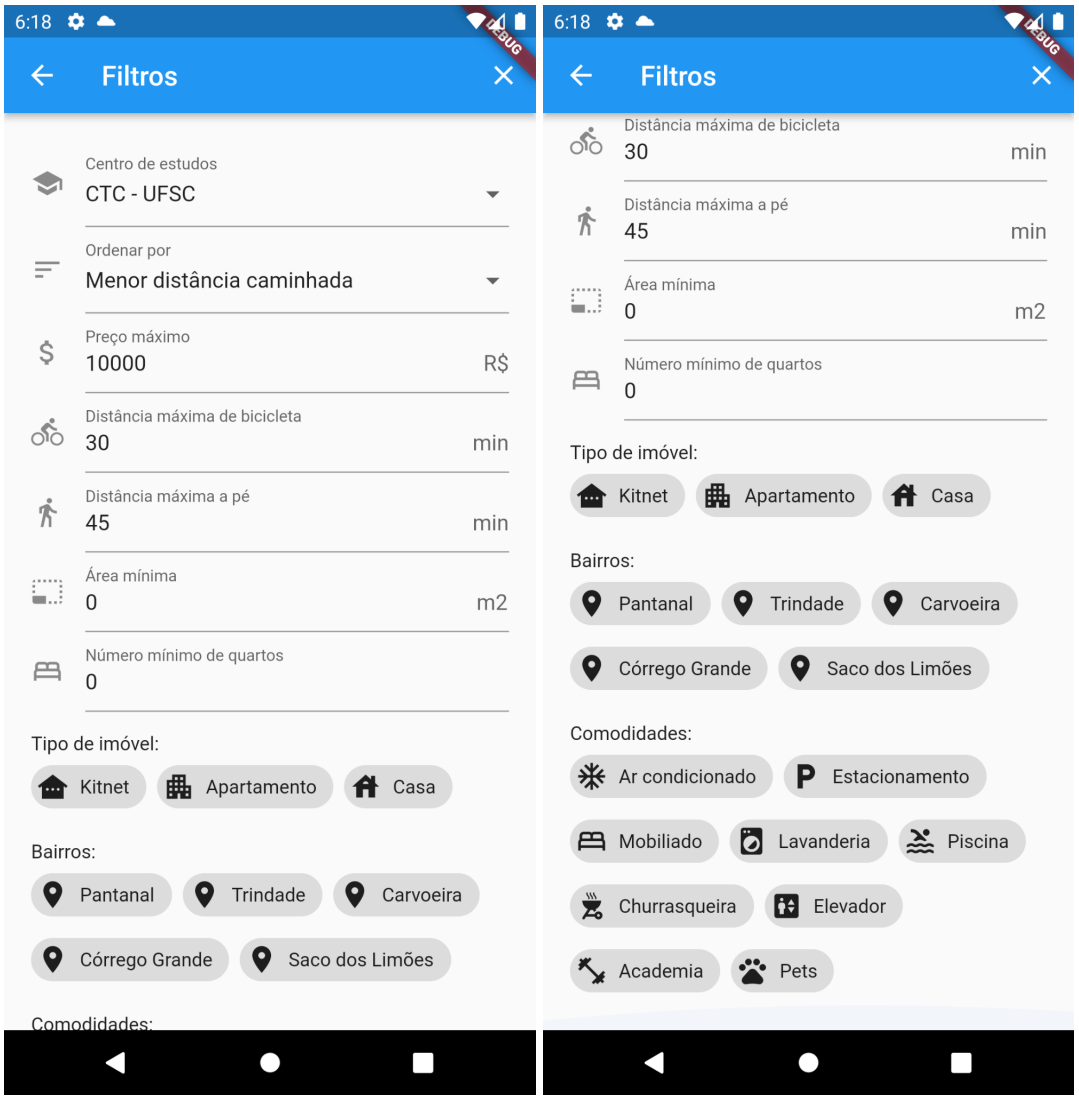
Fonte: O autor (2022)

O usuário também pode clicar em um marcador no mapa, que o leva para uma página com a descrição do imóvel. A barra inferior representa a barra de navegação para as principais telas do aplicativo: mapa, lista e favoritos.

Os filtros são a parte mais importante do aplicativo, que permite ao usuário filtrar os anúncios com as suas preferências pessoais e necessidades. A tela de filtros, como mostra a figura 32, consiste em um formulário onde o usuário pode alterar o centro de estudos, escolher a forma de ordenação (para os anúncios em formato de lista), preço máximo, distância máxima de bicicleta e a pé, bairro, área mínima, número mínimo de quartos, tipo de imóvel, e por fim quais comodidades

estão disponíveis, como ar condicionado, lavanderia, piscina, churrasqueira, elevador, estacionamento, academia, se o imóvel está mobiliado e se aceita pets.

Figura 32 - Telas de filtros de anúncios



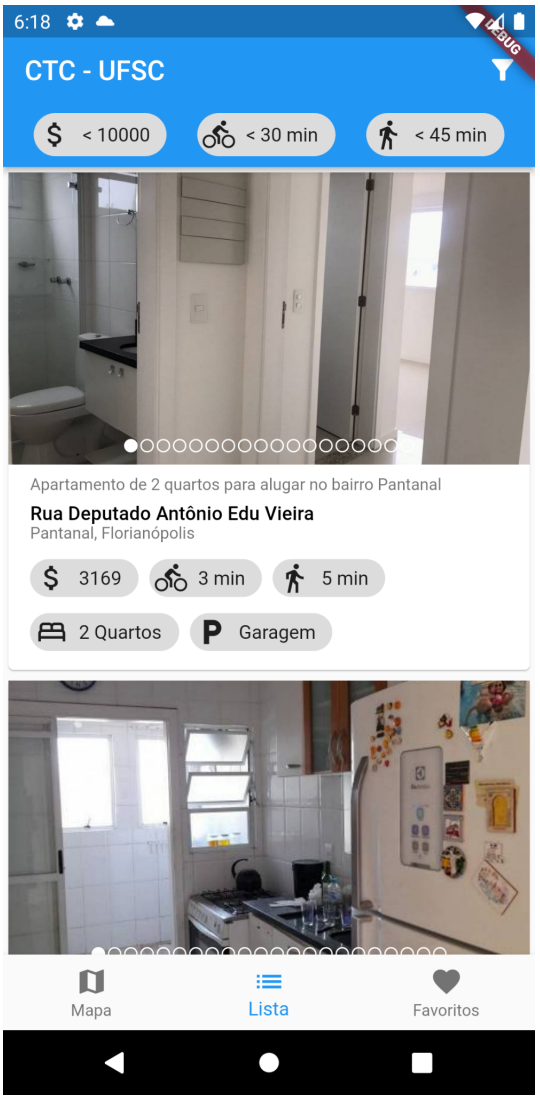
Fonte: O autor (2022)

No canto superior direito da tela de filtros, o usuário pode clicar no ícone “X”, o que fará com que o formulário seja limpo e volte aos valores iniciais definidos pelo desenvolvedor.

Ao selecionar os filtros desejados e voltar para a tela de mapa ou lista, a quantidade de anúncios será alterada, levando em consideração os filtros selecionados, mostrando apenas os anúncios que satisfazem os filtros selecionados.

Caso o usuário deseje ver os anúncios em formato de lista, visualizando as fotos dos anúncios de forma mais compacta, há a tela de lista de anúncios, como mostra a figura 33.

Figura 33 - Tela de listagem de anúncios



Fonte: O autor (2022)

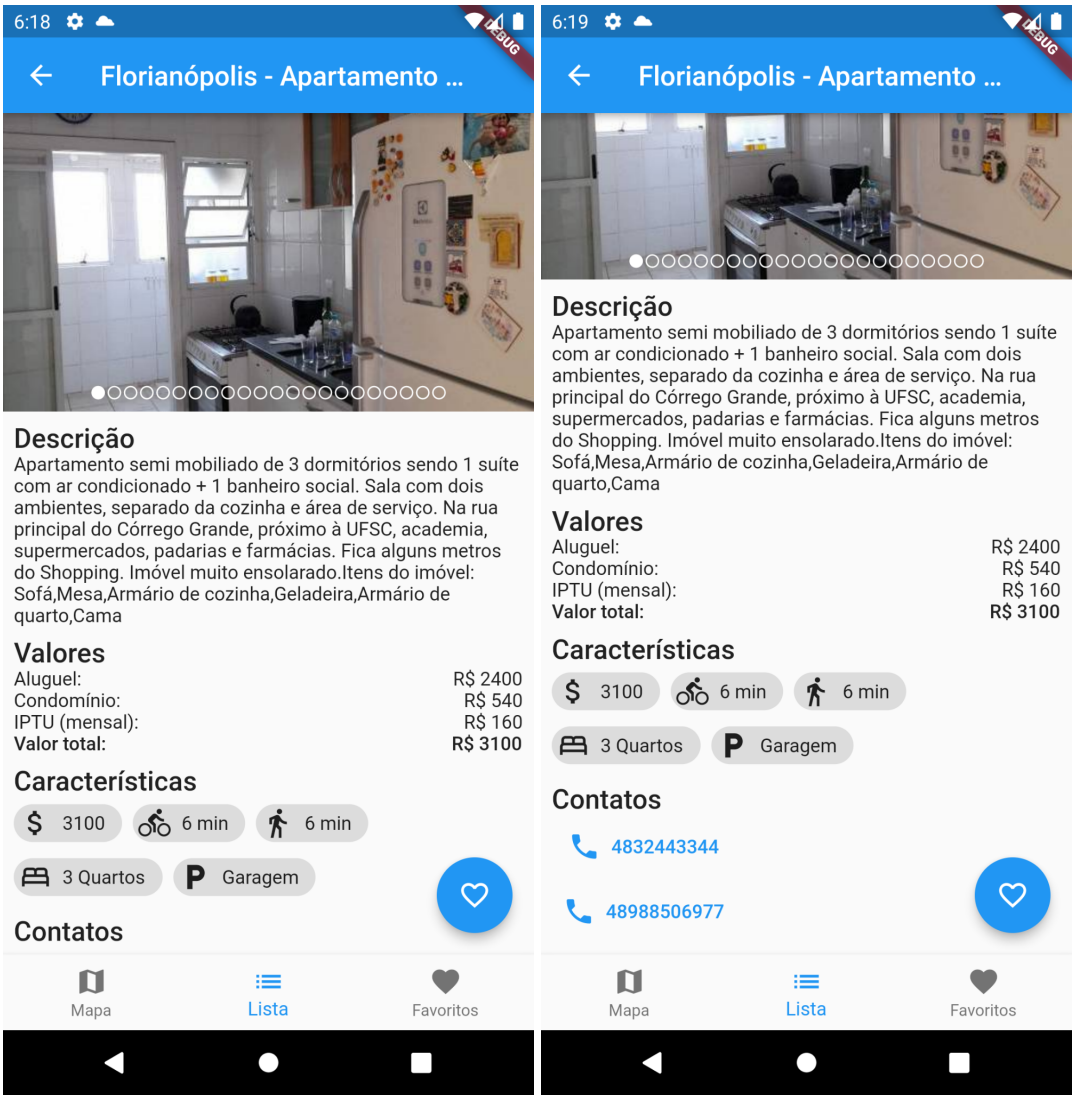
Nesta tela o usuário pode conferir todos os anúncios, bastando apenas rolar a tela para baixo. Também é possível visualizar as imagens de cada anúncio arrastando as imagens para o lado. As bolinhas que sobrepõem as imagens na parte inferior representam a quantidade de imagens no anúncio, sendo que a bolinha cheia indica a imagem que o usuário está visualizando.

Além disso, na tela de lista de anúncios, cada anúncio contém uma breve descrição, título, bairro e por fim são apresentados balões com as principais

características do imóvel. No caso do imóvel da figura acima, são apresentados o valor, as distâncias até o centro de estudos escolhido e as comodidades disponíveis.

Ao achar algum anúncio da lista interessante, o usuário pode clicá-lo, o que o leva para a tela de detalhes do anúncio. Esta mesma tela pode ser aberta pelo usuário através do mapa, ao clicar em um *pin*. Esta tela é representada pela figura 34.

Figura 34 - Telas visualização de anúncio



Fonte: O autor (2022)

Ao entrar nesta tela de detalhamento, o usuário tem acesso às imagens da mesma forma que na lista. Esta tela também contém uma descrição completa do

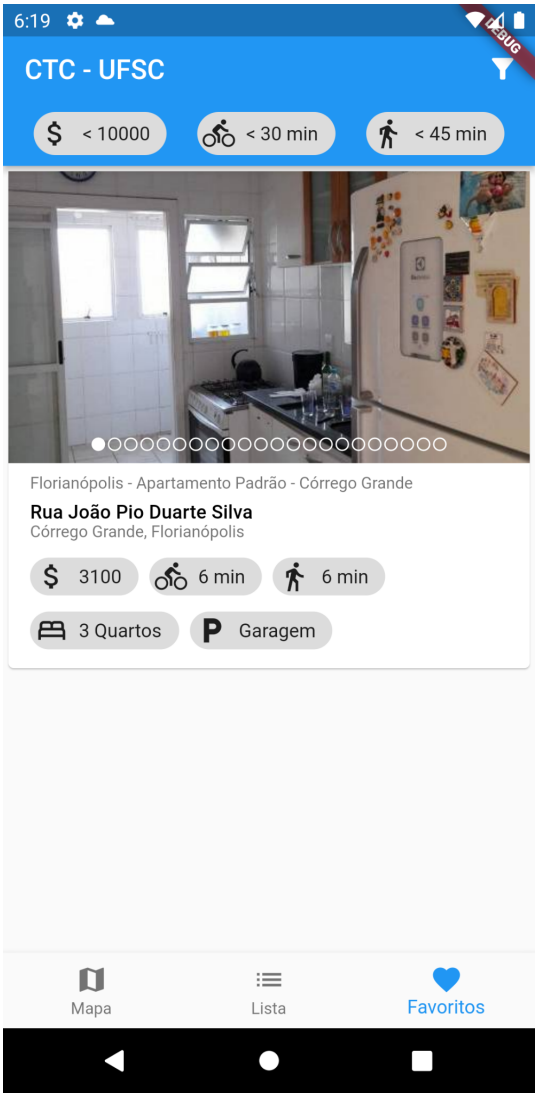
anúncio, a qual foi escrita pelo locador, assim como os valores, comodidades e contato.

Os contatos são botões que, ao serem clicados, abrem o aplicativo de telefone padrão do dispositivo com o número indicado, assim facilitando o contato com o locador.

Ainda nesta tela, há a possibilidade de adicionar o anúncio ao favoritos ao clicar no ícone com símbolo de coração no canto inferior direito da tela. Ao fazer isto, o anúncio é adicionado à lista de favoritos, a qual é salva apenas localmente no aplicativo, e que pode ser visualizada na lista de favoritos, que é acessível pela barra de navegação. O anúncio pode ser removido dos favoritos repetindo a ação. Um ícone de coração vazio indica que o anúncio não está nos favoritos, enquanto um ícone cheio indica que está nos favoritos.

A lista de favoritos é representada pela figura 35, funcionando de forma similar à lista de anúncios previamente discutida. A única diferença nessa tela é que os anúncios que aqui aparecem são aqueles que o usuário marcou como favoritos.

Figura 35 - Tela de favoritos



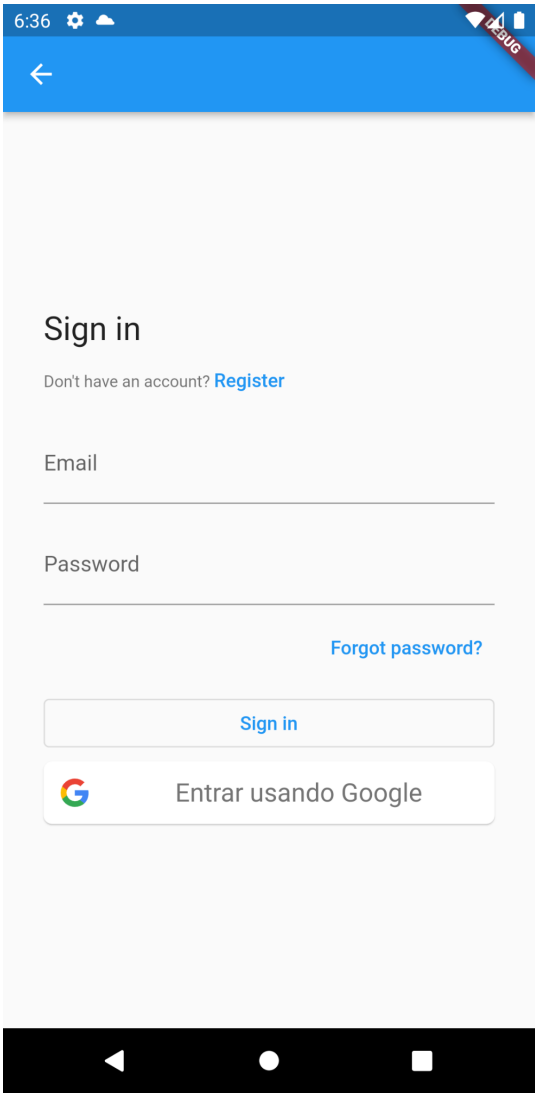
Fonte: O autor (2022)

5.2.2 Locador

Ao clicar no botão "Quero anunciar meu imóvel”, como mostra a figura 30, o usuário é apresentado a uma tela de login, pois é um requisito para os locadores que façam login, assim podendo conectar os anúncios a uma conta específica.

A figura 36 mostra a tela de login mencionada, na qual o usuário pode realizar um cadastro, caso necessário, ou fazer um login usando seu usuário e senha. Para facilitar também é possível realizar o login usando uma conta do Google com o botão “Entrar usando Google”.

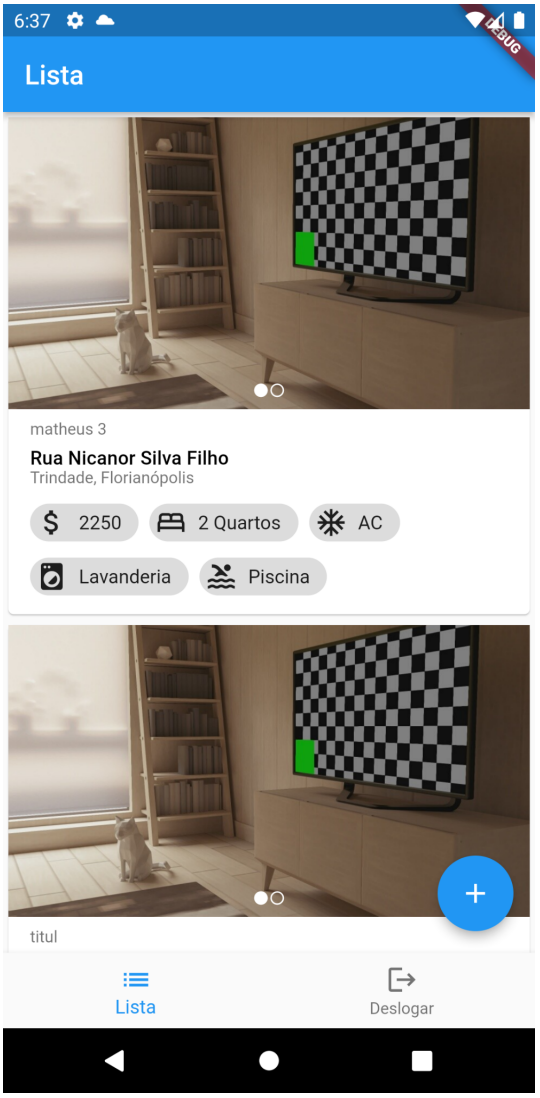
Figura 36 - Tela de login para locador



Fonte: O autor (2022)

Após realizado o login, o usuário é redirecionado para a sua lista de anúncios já cadastrados, como mostra a figura 37. Nesta tela o usuário visualiza seus anúncios cadastrados de forma semelhante ao que é visto pelos locatários. Para adicionar um novo anúncio basta clicar no botão redondo na parte inferior direita.

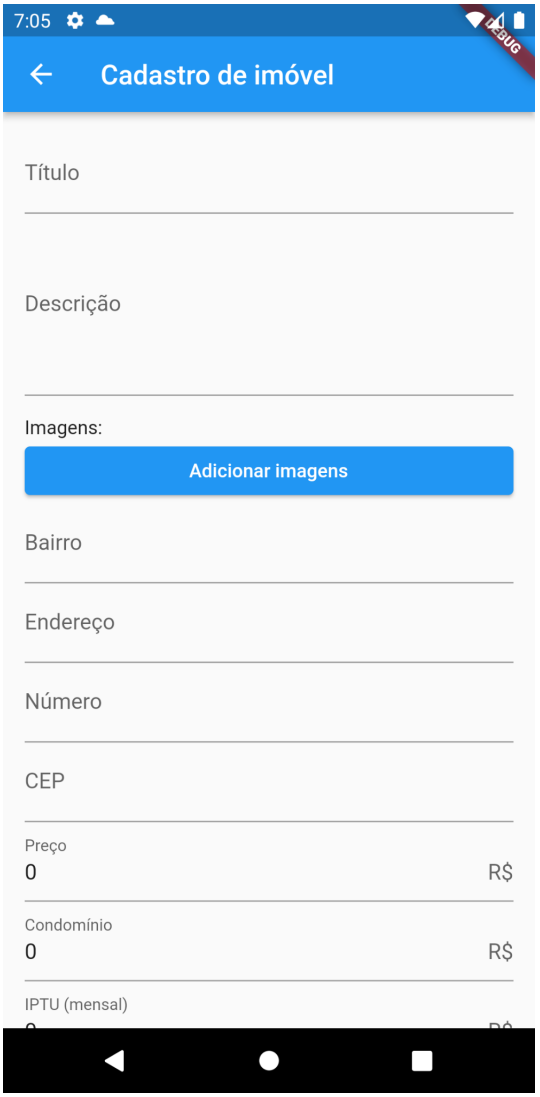
Figura 37 - Tela de listagem de anúncio para locador



Fonte: O autor (2022)

A tela de cadastro de anúncio é um formulário simples onde o locador deve preencher os campos com as informações relevantes sobre o imóvel e adicionar imagens, como pode ser visto na figura 38.

Figura 38 - Tela de cadastro de anúncio



Fonte: O autor (2022)

O locador também pode editar ou remover um anúncio, para isso basta clicar no anúncio desejado na lista de anúncios, conforme a figura 35 mostra. Isto o levará para uma tela de detalhamento um tanto parecida com a tela de detalhamento do locatário. As diferenças são que não é mostrada a distância entre o centro de estudos e o imóvel, pois nenhum centro de estudos é selecionado pelo locador. No canto superior direito há dois ícones, um para editar o anúncio, que leva o usuário para o formulário mostrado pela figura 36, porém já com as informações preenchidas, e o botão para deletar o anúncio, que é representado pelo ícone da lixeira, como mostra a figura 39.

Figura 39 - Tela detalhada de anúncio

Fonte: O autor (2022)

6. AVALIAÇÕES DOS USUÁRIOS

Para validar o conceito proposto pelo aplicativo e receber avaliações da experiência de uso do aplicativo foi elaborado um questionário utilizando o *Google Forms*.

Para que fosse possível instalar o aplicativo de forma fácil para a avaliação, o aplicativo foi enviado para a *Play Store* do *Android* como um aplicativo em fase de beta aberto, ou seja, qualquer usuário com o link para a página do aplicativo pode baixá-lo, como mostra a figura 40.

Figura 40 - Aplicativo listado na *Play Store*

Fonte: O Autor

O questionário foi enviado para cerca de 150 alunos dos cursos de Sistemas de Informação e Ciência da Computação da UFSC, sendo cerca de 110 alunos nas fases finais do curso e 40 alunos nas fases iniciais. As respostas foram coletadas

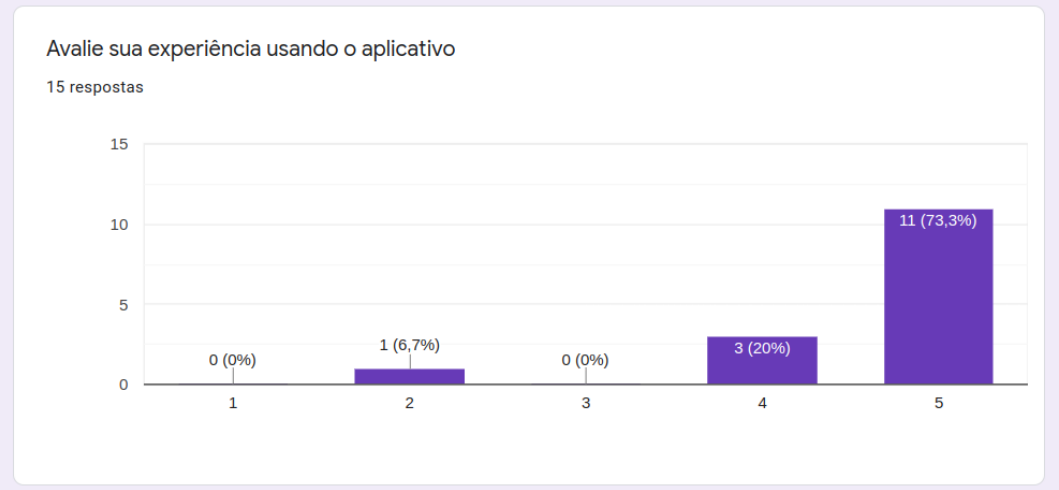
durante o período de 10 dias, sendo o formulário aberto para respostas no dia 3 de fevereiro de 2022 e fechado em 12 de fevereiro de 2022. No total 15 alunos responderam o questionário, tendo assim uma taxa de resposta de 10%.

6.1 Questionário de Avaliação do Aplicativo

As perguntas utilizadas no questionário são descritas a seguir, juntamente com os resultados.

A primeira pergunta do questionário é **“Avalie sua experiência usando o aplicativo"**, tendo como opções de resposta notas de 1 até 5, sendo 1 muito insatisfeito e 5 muito satisfeito. A figura 41 representa as respostas submetidas. No total de 15 respostas, 11 tiveram nota 5, 3 tiveram nota 4 e 1 teve nota 2.

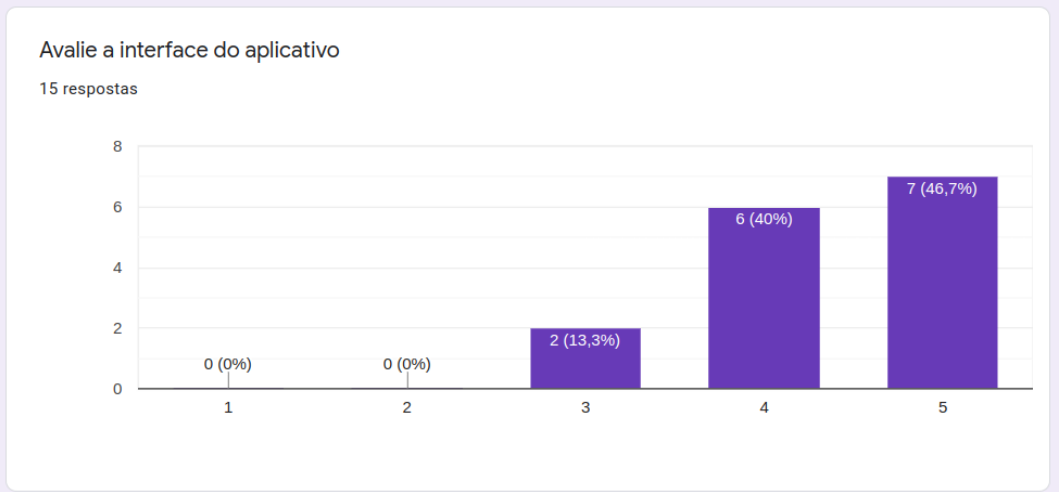
Figura 41 - Resultados de “Avalie sua experiência usando o aplicativo"



Fonte: O Autor

A segunda pergunta do questionário é **“Avalie a interface do aplicativo"**, tendo como opções de resposta notas de 1 até 5, sendo 1 muito insatisfeito e 5 muito satisfeito. A figura 42 representa as respostas submetidas. No total de 15 respostas, 7 tiveram nota 5, 6 tiveram nota 4 e 2 tiveram nota 3.

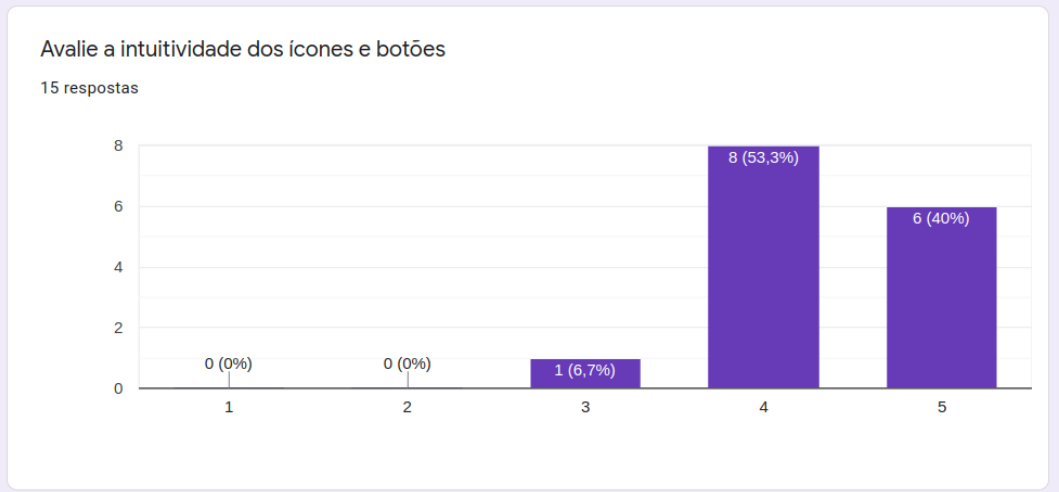
Figura 42 - Resultados de “Avalie a interface do aplicativo"



Fonte: O autor

A terceira pergunta do questionário é “**Avalie a intuitividade dos ícones e botões"**, tendo como opções de resposta notas de 1 até 5, sendo 1 muito insatisfeito e 5 muito satisfeito. A figura 43 representa as respostas submetidas. No total de 15 respostas, 4 tiveram nota 5, 8 tiveram nota 4 e 1 tiveram nota 3.

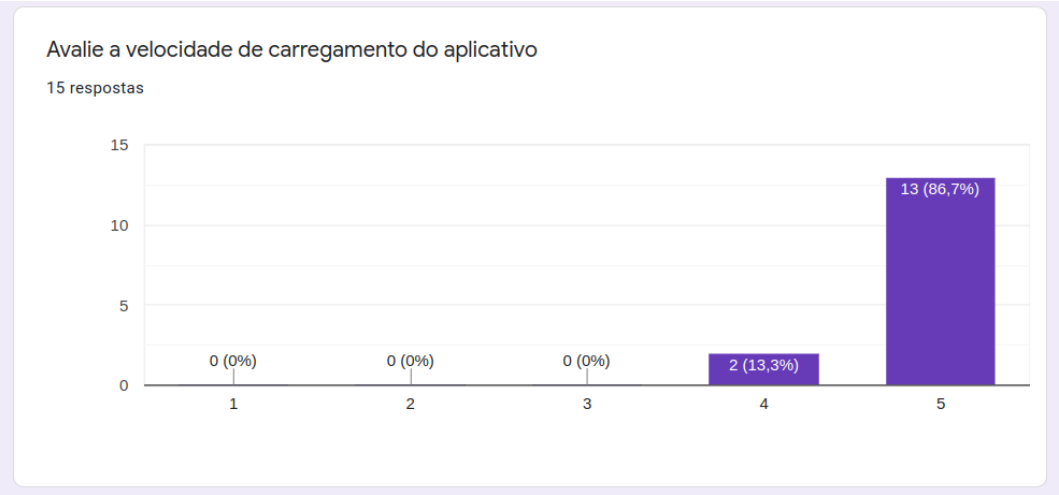
Figura 43 - Resultados de “Avalie a intuitividade dos ícones e botões"



Fonte: O autor

A quarta pergunta do questionário é **“Avalie a velocidade de carregamento do aplicativo"**, tendo como opções de resposta notas de 1 até 5, sendo 1 muito insatisfeito e 5 muito satisfeito. A figura 44 representa as respostas submetidas. No total de 15 respostas, 13 tiveram nota 5 e 2 tiveram nota 4.

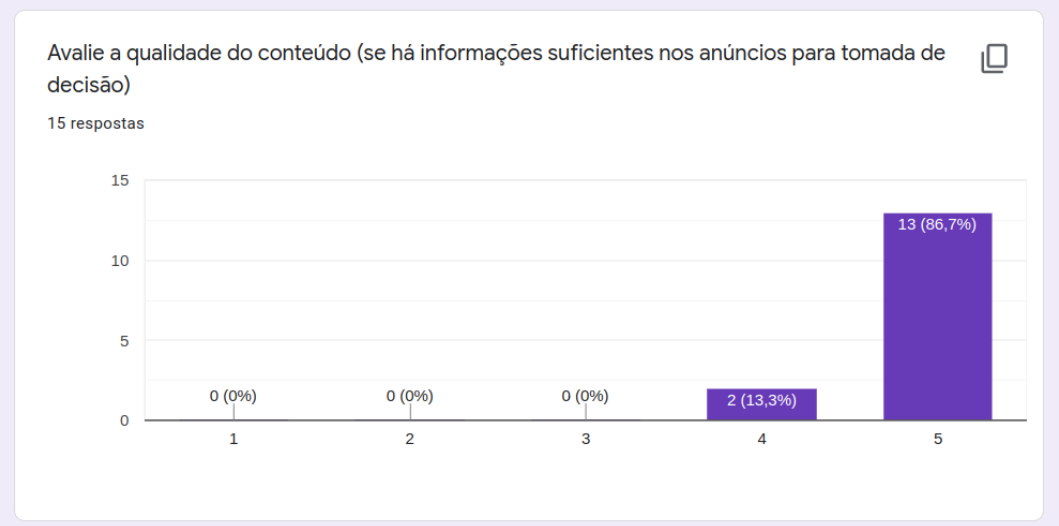
Figura 44 - Resultados de “Avalie a velocidade de carregamento do aplicativo"



Fonte: O autor

A quinta pergunta do questionário é **“Avalie a qualidade do conteúdo (se há informações suficientes nos anúncios para tomada de decisão)"**, tendo como opções de resposta notas de 1 até 5, sendo 1 muito insatisfeito e 5 muito satisfeito. A figura 45 representa as respostas submetidas. No total de 15 respostas, 13 tiveram nota 5 e 2 tiveram nota 4.

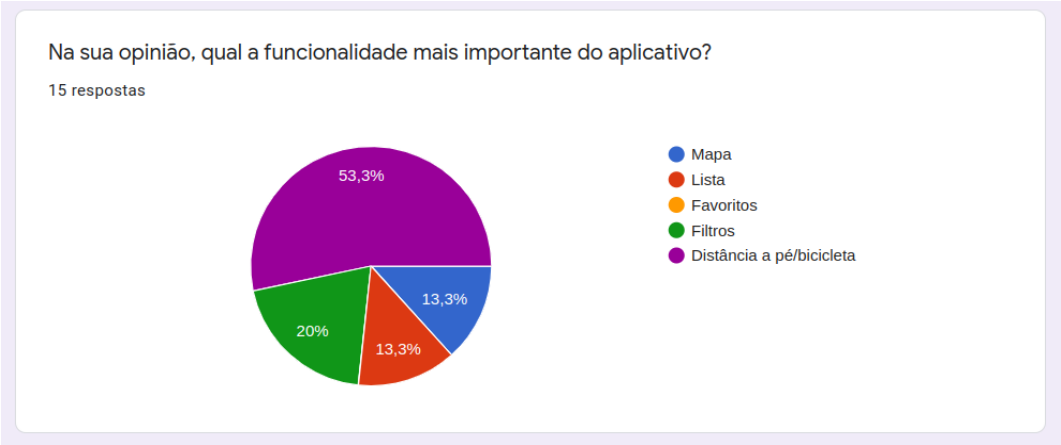
Figura 45 - Resultados de “Avalie a qualidade do conteúdo"



Fonte: O autor

A sexta pergunta do questionário é **“Na sua opinião, qual a funcionalidade mais importante do aplicativo?"**, tendo como opções de resposta uma lista com as seguintes opções: Mapa, Lista, Favoritos, Filtros e Distância a pé/bicicleta. A figura 46 representa as respostas submetidas. No total de 15 respostas, 8 foram Distância a pé/bicicleta, 3 foram Filtros, 2 foram Lista e 2 foram Mapa.

Figura 46 - Resultados de “Qual a funcionalidade mais importante do aplicativo"



Fonte: O autor

A sétima pergunta do questionário é **“Na sua opinião, o aplicativo cumpre seu objetivo?"**, tendo como opções de resposta uma lista com as seguintes opções: Sim, Cumpre parcialmente e Não. A figura 47 representa as respostas submetidas. No total de 15 respostas, 13 foram Sim, e 2 foram Cumpre parcialmente.

Figura 47 - Resultados de “O aplicativo cumpre seu objetivo"

Fonte: O autor

A oitava pergunta do questionário é **“Há alguma funcionalidade que você considera necessária, mas está faltando no aplicativo?"**, tendo como opções de resposta uma lista com as seguintes opções: Sim e Não. A figura 48 representa as respostas submetidas. No total de 15 respostas, 11 foram Não, e 4 foram Sim.

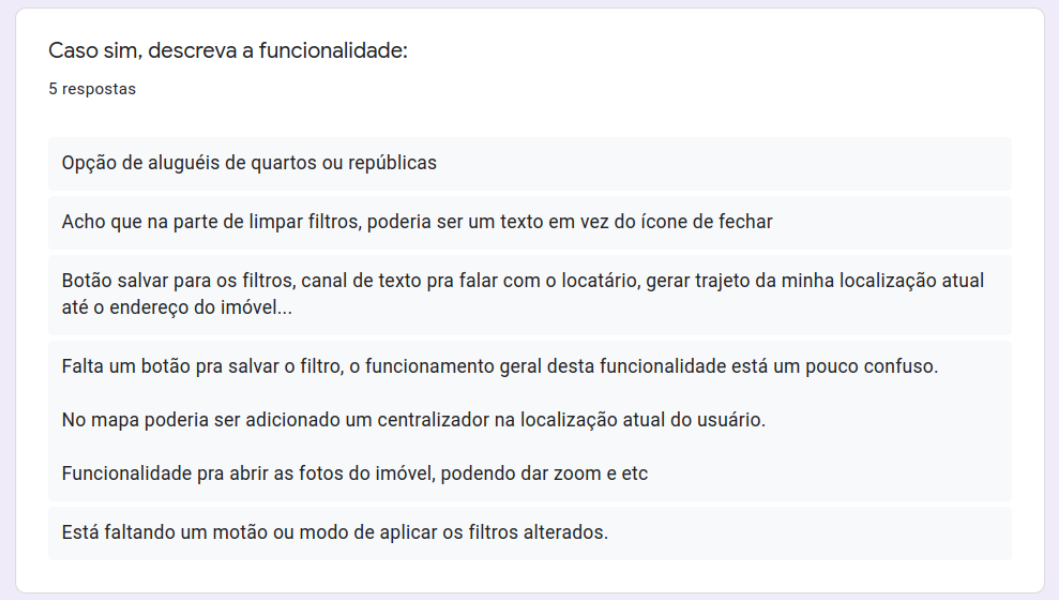
Figura 48 - Resultados de “Há alguma funcionalidade que você considera necessária, mas está faltando no aplicativo?"

Fonte: O autor

A nona e última pergunta do questionário é **“Caso sim (falta alguma funcionalidade), descreva a funcionalidade:"**, tendo como opções de resposta um campo de texto. A figura 49 representa as respostas submetidas. As 5 respostas submetidas foram:

1. “Opção de aluguéis de quartos ou repúblicas”
2. “Acho que na parte de limpar filtros, poderia ser um texto em vez do ícone de fechar”
3. “Botão salvar para os filtros, canal de texto pra falar com o locatário, gerar trajeto da minha localização atual até o endereço do imóvel…”
4. “Falta um botão pra salvar o filtro, o funcionamento geral desta funcionalidade está um pouco confuso.”
5. “No mapa poderia ser adicionado um centralizador na localização atual do usuário.”
6. “Funcionalidade pra abrir as fotos do imóvel, podendo dar zoom e etc”
7. “Está faltando um botão ou modo de aplicar os filtros alterados.”

Figura 49 - Resultados de “**Caso sim (falta alguma funcionalidade), descreva a funcionalidade"**



Fonte: O autor

6.2 Análise dos Resultados

Em geral, o aplicativo foi bem recebido pelos participantes da pesquisa. Nas cinco primeiras perguntas, grande parte das notas estava entre 4 e 5, correspondentes às avaliações *Satisfeito* e *Muito satisfeito*. Além disso, na sétima pergunta “**Na sua opinião, o aplicativo cumpre seu objetivo?**”, 86.7% dos participantes responderam *Sim*, e os demais responderam *Cumpre parcialmente.*

Três dos sete comentários sobre as funcionalidades do aplicativo foram sobre o mesmo item, falta de botão para aplicar os filtros, na tela apresentada na figura 30. Na verdade, estes filtros são aplicados de forma automática, ou seja, ao clicar no botão de voltar os filtros modificados já estão aplicados, mas não foi entendido desta forma por alguns usuários e fica como um ponto de melhora para trabalhos futuros. Logo, isto deveria ser tratado em um trabalho futuro como uma melhoria de interface.

7. CONCLUSÕES

A motivação para o desenvolvimento deste trabalho se deu pelo fato de não haver nenhum aplicativo móvel de anúncio de imóveis para universitários onde é possível selecionar imóveis a partir da distância em minutos entre o imóvel e o centro de estudos.

Durante as várias etapas do trabalho, foram feitas pesquisas de mercado para entender o ambiente de aplicativos de anúncio de imóveis, após isso foi realizada a proposta da solução com casos de uso e wireframes das telas, e por fim foi realizado o desenvolvimento do aplicativo para Android e realizada a pesquisa com os usuários.

Com os resultados obtidos, foi possível definir que o aplicativo foi bem recebido e existe valor no uso dele para os alunos da universidade, especialmente pelas respostas da pergunta **“Na sua opinião, o aplicativo cumpre seu objetivo?"**, onde mais de 85% responderam que **Sim**, e o restante **Cumpre parcialmente.**

8. TRABALHOS FUTUROS

Existem múltiplos trabalhos futuros para o aplicativo. De acordo com o *feedback* dos usuários, existem vários pontos de melhora na parte visual e de experiência do usuário, conforme mostra a análise dos resultados.

Os outros comentários deixados pelos usuários na última resposta do questionário também servem como trabalhos futuros, como por exemplo adicionar a funcionalidade de procurar por repúblicas ou quartos, dar zoom nas fotos, mostrar localização do usuário.

Além disso, também poderia ser feita uma coleta de dados dos usuários para otimizar os anúncios, como por exemplo em quais anúncios tal usuário clicou e por quanto tempo ficou neles, e junto com os dados fornecidos pelo usuário poderiam ser geradas estatísticas que forneceriam dados relevantes para os locadores.

E para os locadores poderia ser implementado uma visualização de estatísticas dos anúncios, mostrando quantos visualizações o anúncio possui, qual perfil de usuário visualiza os anúncios, etc.

9. REFERÊNCIAS BIBLIOGRÁFICAS

ALI, Yusuf. **Rent a home: a cross platform mobile application to list and search rental homes.** Kansas State University. 2016.

BIØRN-HANSEN, Andreas; GRØNLI, Tor-Morten; GHINE, Gheorghita; ALOUNEH, Sahel. **An Empirical Study of Cross-Platform Mobile Development in Industry**. 2019.

WU, Wenhao. **React Native vs Flutter, cross-platform mobile application frameworks.** Metropolia University of Applied Sciences. 2018.

DART. **The new AdWords UI uses Dart — we asked why.** 2016. Disponível em: https://news.dartlang.org/2016/03/the-new-adwords-ui-uses-dart-we-asked.html. Acesso em: 03 Set. 2021.

GOOGLE. **Start thinking declaratively.** 2021. Disponível em: https://flutter.dev/docs/development/data-and-backend/state-mgmt/declarative. Acesso em: 03 Set. 2021.

AGILE ALLIANCE. **Agile 101.** 2021. Disponível em: https://www.agilealliance.org/agile101/. Acesso em: 17 Set. 2021.

GOOGLE. **Inside Flutter.** 2021. Disponível em: https://flutter.dev/docs/resources/inside-flutter. Acesso em: 03 Set. 2021.

GOOGLE. **Flutter architectural overview.** 2021. Disponível em: https://flutter.dev/docs/resources/architectural-overview. Acesso em: 03 Set. 2021.

GOOGLE. **Introduction to widgets.** 2021. Disponível em: https://flutter.dev/docs/development/ui/widgets-intro. Acesso em: 03 Set. 2021.

TILLU, Jay. **What is a Widget in Flutter? Let's clear the basics first.** 2021. Disponível em: https://medium.com/jay-tillu/4-what-is-widget-in-flutter-lets-clear-the-basics-first-82f501c8d0f0. Acesso em: 01 Set. 2021.

GOOGLE. **Hot reload.** 2021. Disponível em: https://flutter.dev/docs/development/tools/hot-reload. Acesso em: 05 Set. 2021.

YEGULALP, Serdar. **What is LLVM? The power behind Swift, Rust, Clang, and more.** 2020. Disponível em: https://www.infoworld.com/article/3247799/what-is-llvm-the-power-behind-swift-rust-clang-and-more.html. Acesso em: 05 Set. 2021.

GOOGLE. **Android NDK.** 2021. Disponível em: https://developer.android.com/ndk. Acesso em: 05 Set. 2021.

MROCZKOWSKA, Agnieszka. **What Is a Mobile App?** 2021. Disponível em: https://www.thedroidsonroids.com/blog/what-is-a-mobile-app-app-development-basics-for-businesses. Acesso em: 01 Set. 2021.

GOOGLE. **Distance Matrix API Overview.** 2022. Disponível em: https://developers.google.com/maps/documentation/distance-matrix/overview. Acesso em: 31 Jan. 2022.

GOOGLE. **Geocoding API Overview.** 2022. Disponível em: https://developers.google.com/maps/documentation/geocoding/overview Acesso em: 31 Jan. 2022.

GOOGLE. **Cloud Functions for Firebase.** 2022. Disponível em: https://firebase.google.com/docs/functions Acesso em: 31 Jan. 2022.

GOOGLE. **Firebase Authentication.** 2022. Disponível em: https://firebase.google.com/docs/auth Acesso em: 31 Jan. 2022.

GOOGLE. **Cloud Firestore.** 2022. Disponível em: https://firebase.google.com/docs/firestore Acesso em: 31 Jan. 2022.

GOOGLE. **Cloud Storage for Firebase.** 2022. Disponível em: https://firebase.google.com/docs/storage Acesso em: 31 Jan. 2022.

IBM. **REST APIs.** 2022. Disponível em: https://www.ibm.com/cloud/learn/rest-apis Acesso em: 17 Fev. 2022.

BALSAMIQ, **What Are Wireframes?**. 2022. Disponível em: https://balsamiq.com/learn/articles/what-are-wireframes/. Acesso em: 13 Mar. 2022.

APÊNDICE A - Artigo

Aplicativo móvel para busca de imóveis para universitários

Matheus Andrade Alves [1], Frank Augusto Siqueira [1]

[1]Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis / SC, Brasil

matheus.aa@grad.ufsc.br, frank.siqueira@ufsc.br

Abstract. *There are currently some websites that offer services for landlords to advertise their properties specifically for college students. However, these sites don't have a good usability and don't make good use of a filter system taking into account the needs of a university student who wants to live close to his college and wants to find an accommodation that fits his budget and that has the features that meet your needs. The objective of this work is to develop a mobile application on Android and iOS platforms for university students to search for accommodation, for example real estate, dormitories or rooms through a map, using specific filters for university students. For this, software engineering methods are used to define the product and the software architecture. At the end of the work, a survey is carried out with the students to validate the application, its usability and functionalities, which returns positive results and validates the objectives of the work..*

Resumo. *Atualmente existem alguns sites que oferecem serviços para locadores anunciarem seus imóveis especificamente para universitários. Porém, estes sites não possuem uma usabilidade boa e não fazem um bom uso de um sistema de filtros levando em conta as necessidades de um estudante universitário que deseja morar perto de sua faculdade e queira encontrar uma acomodação que encaixe no seu orçamento e que tenha as características que atendem às suas necessidades. O objetivo deste trabalho é desenvolver um aplicativo móvel nas plataformas Android e iOS para alunos de universidades procurarem acomodações, por exemplo imóveis, repúblicas ou quartos por meio de um mapa, utilizando filtros específicos para universitários. Para isso são utilizados métodos de engenharia de software para definição do produto e da arquitetura do software. Ao final do trabalho, é realizada uma pesquisa com os alunos para validar o aplicativo, sua usabilidade e funcionalidades, a qual retorna resultados positivos e valida os objetivos do trabalho..*

1. Introdução

Atualmente não existem opções de aplicativos de busca de imóveis para aluguel especificamente voltados às necessidades específicas de alunos universitários, segundo a pesquisa realizada por este trabalho. O aluno que deseja procurar um imóvel perto da UFSC tem a opção de procurar um imóvel em um site ou aplicativo de corretoras de imóveis, as quais nem sempre têm em mente este público alvo e não possuem ferramentas específicas para filtrar os imóveis de acordo com as necessidades de alunos de universidades. Além dos sites de corretoras, existe o site de classificados da UFSC, disponível em https://classificados.inf.ufsc.br, onde há uma seção de classificados para imóveis, no qual aparentemente a maioria dos alunos procura por um lugar para morar perto da universidade. Porém, este site peca na usabilidade e não possibilita nenhum tipo de filtro nas buscas, dificultando bastante a procura por um imóvel.

Tendo este problema em mente, este trabalho propõe o desenvolvimento de um aplicativo móvel específico para alunos universitários que desejam procurar uma moradia perto da universidade. Para isso, neste trabalho será pesquisado o estado da arte nos atuais aplicativos de imóveis para entender quais funcionalidades são mais úteis e desejadas pelos usuários, e também serão investigados os atuais aplicativos específicos para universitários para entender quais são os itens mais procurados pelos alunos ao procurar por um imóvel.

Com estes dados será desenvolvido um aplicativo móvel para Android e iOS que terá os anúncios de imóveis disponibilizados em um mapa, onde os imóveis serão representados por marcadores. Os usuários que procuram um imóvel poderão inserir algumas informações pessoais como: curso, centro, meio de transporte que será utilizado para locomoção até a faculdade, entre outros, para ajudar na busca de um imóvel. Além disso, serão desenvolvidos filtros específicos para o público alvo baseado nos estudos anteriores, em específico um filtro para a distância entre o imóvel e o centro de estudos do aluno, de modo que o aluno possa filtrar anúncios por tempo máximo de deslocamento entre o imóvel e o centro de estudos.

Para otimizar os anúncios, o aplicativo irá coletar dados dos usuários, como por exemplo em quais anúncios tal usuário clicou e por quanto tempo ficou neles, e junto com os dados fornecidos pelo usuário serão geradas estatísticas que fornecerão dados para os locadores.

Espera-se que com o aplicativo, os alunos consigam procurar com facilidade imóveis para alugar perto da sua universidade e que atendam às suas necessidades para estudos. Para validar esse objetivo será realizada uma avaliação com os usuários ao fim do projeto a fim de receber feedback sobre o aplicativo, suas funcionalidades e usabilidade.

2. Estado da arte

Com finalidade de identificar o estado atual em que se encontram os aplicativos comerciais e não-comerciais relacionados a anúncios de imóveis, voltados a universitários ou não, foram encontradas através de buscas pelo Google durante o mês de agosto de 2021, utilizando como palavras-chave: aluguel, ufsc, imóveis, Florianópolis, universidade.

Percebe-se que nos serviços voltados para universitários como Classificados UFSC, Federal SC e Morar USP faltam muitos filtros úteis para os imóveis comparado com Quinto Andar, VivaReal e Realtor. Fica clara a diferença tecnológica entre as soluções voltadas para universitários e as voltadas para o mercado geral.

O único serviço que provê algum filtro de distância é o Realtor. Não foi possível encontrar nenhum filtro parecido nos serviços brasileiros pesquisados. Quase todos os serviços têm um ponto positivo, mas não há uma junção de todos estes pontos em um só aplicativo, ainda mais para o público universitário, conforme a tabela abaixo retrata.

Tabela 1 - Comparativo entre os serviços

Serviço	Público	Filtros avançados	Possui Mapa	Filtro por distância	Possui App
Classificados UFSC	Universitários	Não	Não	Não	Não
Federal SC	Universitários	Não	Não	Não	Não
Morar USP	Universitários	Não	Sim	Não	Não
QuintoAndar	Geral	Sim	Sim	Não	Sim
VivaReal	Geral	Sim	Sim	Não	Sim
Realtor	Geral	Sim	Sim	Sim	Sim

Fonte: O autor (2021)

3. Proposta

Este projeto propõe criar um aplicativo móvel de anúncio de imóveis focado para o público universitário. Primeiramente o usuário escolherá em qual centro ele estuda na universidade, por exemplo: Centro Tecnológico da Universidade Federal de Santa Catarina. Com isso, ele irá visualizar os imóveis próximos da região do centro, tanto no formato de mapa, onde os imóveis aparecerão como pontos no mapa, quanto no formato de lista. Os anúncios poderão ser filtrados utilizando-se os filtros já conhecidos de outras ferramentas. Além disso, será possível filtrar pela distância em minutos a pé, de bicicleta, carro e transporte público entre o imóvel e o centro de estudo. Ao clicar em um anúncio o usuário poderá ver as informações necessárias, como descrição, fotos, características e contato.

Para que existam anúncios no aplicativo, também serão desenvolvidas funcionalidades para o locatário, onde este usuário poderá cadastrar e gerenciar seus anúncios.

4. Desenvolvimento

O projeto consiste em duas grandes partes, um cliente e um servidor. O servidor será um serviço remoto responsável por armazenar informações sobre os usuários e seus anúncios. A principal responsabilidade deste servidor será receber as requisições de

cadastro de anúncios e calcular a distância entre o imóvel da requisição e os centros cadastrados da universidade, e armazenar o anúncio junto do cálculo. Além disso, ele irá prover algumas outras funcionalidades necessárias para criação, leitura e edição dos anúncios. Essas requisições serão feitas através de uma interface REST API.

A parte cliente será capaz de consumir os dados através da interface mencionada, lendo e enviando os dados necessários. Será responsabilidade do cliente mostrar tais dados para o usuário através de uma interface gráfica de forma organizada, também será responsabilidade do cliente prover filtros para que os dados sejam filtrados na interface gráfica.

5. Conclusão

A motivação para o desenvolvimento deste trabalho se deu pelo fato de não haver nenhum aplicativo móvel de anúncio de imóveis para universitários onde é possível selecionar imóveis a partir da distância em minutos entre o imóvel e o centro de estudos.

Durante as várias etapas do trabalho, foram feitas pesquisas de mercado para entender o ambiente de aplicativos de anúncio de imóveis, após isso foi realizada a proposta da solução com casos de uso e wireframes das telas, e por fim foi realizado o desenvolvimento do aplicativo para Android e realizada a pesquisa com os usuários.

Com os resultados obtidos, foi possível definir que o aplicativo foi bem recebido e existe valor no uso dele para os alunos da universidade, especialmente pelas respostas da pergunta “Na sua opinião, o aplicativo cumpre seu objetivo?", onde mais de 85% responderam que Sim, e o restante Cumpre parcialmente.

Referências

IBM. REST APIs. 2022. Disponível em: https://www.ibm.com/cloud/learn/rest-apis Acesso em: 17 Fev. 2022.

GOOGLE. Distance Matrix API Overview. 2022. Disponível em: https://developers.google.com/maps/documentation/distance-matrix/overview. Acesso em: 31 Jan. 2022.

GOOGLE. Geocoding API Overview. 2022. Disponível em: https://developers.google.com/maps/documentation/geocoding/overview Acesso em: 31 Jan. 2022.

GOOGLE. Cloud Functions for Firebase. 2022. Disponível em: https://firebase.google.com/docs/functions Acesso em: 31 Jan. 2022.

GOOGLE. Firebase Authentication. 2022. Disponível em: https://firebase.google.com/docs/auth Acesso em: 31 Jan. 2022.

GOOGLE. Cloud Firestore. 2022. Disponível em: https://firebase.google.com/docs/firestore Acesso em: 31 Jan. 2022.

GOOGLE. Cloud Storage for Firebase. 2022. Disponível em: https://firebase.google.com/docs/storage Acesso em: 31 Jan. 2022.

APÊNDICE B - Código fonte

::::::::::::::
core/landlord/listings/listing_form_provider.dart
::::::::::::::

```
import 'dart:async';
import 'dart:io';

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:mobile/core/landlord/listings/listings_provider.dart';
import 'package:mobile/core/landlord/listings/listings_service.dart';
import 'package:mobile/core/models/address_model.dart';
import 'package:mobile/core/models/listing_model.dart';

class LandlordListingFormProvider extends ChangeNotifier {
  LandlordListingFormProvider(
    LandlordListingsProvider listings, {
    Listing? listing,
  }) {
    _service = LandlordListingsService();
    _listings = listings;
    _picturesUris = List<File>.empty(growable: true);

    if (listing != null) {
      id = listing.id;
      title = listing.title;
      description = listing.description;
      // _picturesUris =
      neighborhood = getNeighborgoodString(listing.address.neighborhood);
      address = listing.address.street;
      streetNumber = listing.address.streetNumber;
      zipCode = listing.address.zipCode;
      priceMonthly = listing.priceMonthly;
      condoFeeMonthly = listing.condoFeeMonthly;
      propertyTaxMonthly = listing.propertyTaxMonthly;
```

```
    bedrooms = listing.bedrooms;
    squareMeters = listing.squareMeters;
    parkingSpaces = listing.parkingSpaces;
    phoneNumbers = listing.phoneNumbers[0] as String;
    unitType = listing.unitType;

    hasAirConditioning = listing.hasAirConditioning;
    isFurnished = listing.isFurnished;
    hasLaundry = listing.hasLaundry;
    hasPool = listing.hasPool;
    hasGrill = listing.hasGrill;
    hasElevator = listing.hasElevator;
    hasGym = listing.hasGym;
    petsAllowed = listing.petsAllowed;

    notifyListeners();
  }
}

late LandlordListingsService _service;
late LandlordListingsProvider _listings;
String? id;
int? priceMonthly = 0;
int? condoFeeMonthly = 0;
int? propertyTaxMonthly = 0;
int? squareMeters = 0;
String? title = '';
String? description = '';
String? phoneNumbers = '';
late List<File> _picturesUris;
String? neighborhood = '';
String? address = '';
String? zipCode = '';
String? streetNumber = '';
int? parkingSpaces = 0;
int? bedrooms = 0;
UnitType? unitType;
```

```
List<File> get picturesUris => _picturesUris;
set picturesUris(List<File> value) {
  _picturesUris = value;
  notifyListeners();
}

bool _hasLaundry = false;
bool get hasLaundry => _hasLaundry;
set hasLaundry(bool value) {
  _hasLaundry = value;
  notifyListeners();
}

bool _isFurnished = false;
bool get isFurnished => _isFurnished;
set isFurnished(bool value) {
  _isFurnished = value;
  notifyListeners();
}

bool _hasPool = false;
bool get hasPool => _hasPool;
set hasPool(bool value) {
  _hasPool = value;
  notifyListeners();
}

bool _hasGrill = false;
bool get hasGrill => _hasGrill;
set hasGrill(bool value) {
  _hasGrill = value;
  notifyListeners();
}

bool _hasElevator = false;
bool get hasElevator => _hasElevator;
```

```
 set hasElevator(bool value) {
  _hasElevator = value;
  notifyListeners();
 }

 bool _hasGym = false;
 bool get hasGym => _hasGym;
 set hasGym(bool value) {
  _hasGym = value;
  notifyListeners();
 }

 bool _hasAirConditioning = false;
 bool get hasAirConditioning => _hasAirConditioning;
 set hasAirConditioning(bool value) {
  _hasAirConditioning = value;
  notifyListeners();
 }

 bool _petsAllowed = false;
 bool get petsAllowed => _petsAllowed;
 set petsAllowed(bool value) {
  _petsAllowed = value;
  notifyListeners();
 }

 List<String> getAmenities() {
  final List<String> amenities = List<String>.empty(growable: true);

  if (hasLaundry) {
   amenities.add('LAUNDRY');
  }
  if (isFurnished) {
   amenities.add('FURNISHED');
  }
  if (hasPool) {
   amenities.add('POOL');
```

```
    }
    if (hasGrill) {
      amenities.add('BARBECUE_GRILL');
    }
    if (hasElevator) {
      amenities.add('ELEVATOR');
    }
    if (hasGym) {
      amenities.add('GYM');
    }
    if (hasAirConditioning) {
      amenities.add('AIR CONDITIONING');
    }
    if (petsAllowed) {
      amenities.add('PETS_ALLOWED');
    }

    return amenities;
  }

  Future<void> send() async {
    final String? userId = FirebaseAuth.instance.currentUser?.uid;
    final Map<String, dynamic> data = <String, dynamic>{
      'address': <String, dynamic>{
        'city': 'Florianópolis',
        'country': 'BR',
        'neighborhood': neighborhood,
        'state': 'Santa Catarina',
        'street': address,
        'streetNumber': streetNumber,
        'zipCode': zipCode,
      },
      'amenities': getAmenities(),
      'bedrooms': bedrooms,
      'condoFee': condoFeeMonthly.toString(),
      'iptu': propertyTaxMonthly.toString(),
      'phonenumbers': <String?>[phoneNumbers],
```

```
      'title': title,
      'description': description,
      'price': priceMonthly.toString(),
      'unitType': getStringFromUnitType(unitType!),
      'usableArea': squareMeters.toString(),
      'userId': userId,
    };

    late DocumentReference<Map<String, dynamic>> result;

    if (id != null) {
      result = await _service.update(data, id!);
    } else {
      result = await _service.add(data);
    }

    if (picturesUris.isNotEmpty) {
      final List<String> picturesUrls =
          await _service.uploadImages(picturesUris, result.id.toString());
      await result.set(<String, dynamic>{
        'picturesUri': picturesUrls,
      }, SetOptions(merge: true));
    }

    Timer(const Duration(seconds: 1), () {
      _listings.fetchListings();
    });
  }
}
::::::::::::::
core/landlord/listings/listings_provider.dart
::::::::::::::
import 'package:flutter/foundation.dart';
import 'package:mobile/core/landlord/listings/listings_service.dart';
import 'package:mobile/core/models/listing_model.dart';

class LandlordListingsProvider extends ChangeNotifier {
```

```
  LandlordListingsProvider() {
    _service = LandlordListingsService();

    fetchListings();
  }

  late LandlordListingsService _service;
  bool _isLoading = false;
  List<Listing> _listings = List<Listing>.empty();

  set isLoading(bool isLoading) {
    _isLoading = isLoading;
    notifyListeners();
  }

  bool get isLoading => _isLoading;

  set listings(List<Listing> listings) {
    _listings = listings;
    notifyListeners();
  }

  List<Listing> get listings => _listings;

  Future<void> fetchListings() async {
    isLoading = true;
    listings = await _service.fetchAll();
    isLoading = false;
  }

  Future<void> delete(Listing listing) async {
    await _service.delete(listing);

    return fetchListings();
  }
}
::::::::::::::
```

core/landlord/listings/listings_service.dart

```
import 'dart:io';

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:mobile/core/models/listing_model.dart';

class LandlordListingsService {
  Future<List<Listing>> fetchAll() async {
    final FirebaseFirestore firestore = FirebaseFirestore.instance;

    final String? userId = FirebaseAuth.instance.currentUser?.uid;

    final QuerySnapshot<Map<String, dynamic>> snapshot = await firestore
        .collection('listings')
        .where('userId', isEqualTo: userId)
        .get();

    return snapshot.docs
        .map((QueryDocumentSnapshot<Map<String, dynamic>> document) =>
            Listing.fromDocumentSnapshot(document.data()))
        .toList();
  }

  Future<DocumentReference<Map<String, dynamic>>> add(
      Map<String, dynamic> data) async {
    final FirebaseFirestore firestore = FirebaseFirestore.instance;
    final CollectionReference<Map<String, dynamic>> listings =
        firestore.collection('listings');

    final DocumentReference<Map<String, dynamic>> doc =
        await listings.add(data);
    await listings.doc(doc.id).set(<String, String>{
      'id': doc.id.toString(),
    }, SetOptions(merge: true));
```

```
  return doc;
 }

 Future<DocumentReference<Map<String, dynamic>>> update(
     Map<String, dynamic> data, String id) async {
  final FirebaseFirestore firestore = FirebaseFirestore.instance;
  final CollectionReference<Map<String, dynamic>> listings =
      firestore.collection('listings');

  await listings.doc(id).update(data);

  return listings.doc(id);
 }

 Future<List<String>> uploadImages(List<File> files, String listingId) async {
  final List<String> urls = List<String>.empty(growable: true);

  for (int i = 0; i < files.length; i++) {
   final File file = files[i];

   final String filename =
       file.path.substring(file.path.lastIndexOf('/') + 1);

   final Reference ref =
       FirebaseStorage.instance.ref('$listingId/$filename');
   await ref.putFile(file);
   final String link = await ref.getDownloadURL();
   urls.add(link);
  }

  return urls;
 }

 Future<void> delete(Listing listing) {
  final FirebaseFirestore firestore = FirebaseFirestore.instance;
  final CollectionReference<Map<String, dynamic>> listings =
```

```
        firestore.collection('listings');

    return listings.doc(listing.id).delete();
  }
}
::::::::::::::
core/models/address_model.dart
::::::::::::::
import 'package:google_maps_flutter/google_maps_flutter.dart';

class Address {
  Address({
    required this.country,
    required this.state,
    required this.city,
    required this.neighborhood,
    required this.neighborhoodString,
    required this.street,
    required this.streetNumber,
    required this.zipCode,
    required this.position,
  });

  Address.fromMap(Map<String, dynamic> map)
      : country = map['country'] as String,
        state = map['state'] as String,
        city = map['city'] as String,
        neighborhood = _getNeighborhood(map['neighborhood'] as String),
        neighborhoodString = map['neighborhood'] as String,
        street = map['street'] as String,
        streetNumber = map['streetNumber'] as String,
        zipCode = map['zipCode'] as String,
        position = LatLng(
            map['point']['lat'] as double, map['point']['lon'] as double);

  String country;
  String state;
```

```
  String city;
  Neighborhood neighborhood;
  String neighborhoodString;
  String street;
  String streetNumber;
  String zipCode;
  LatLng position;
}

enum Neighborhood {
  PANTANAL,
  TRINDADE,
  CARVOEIRA,
  SACO_DOS_LIMOES,
  CORREGO_GRANDE,
}

Neighborhood _getNeighborhood(String value) {
  switch (value.toLowerCase()) {
    case 'pantanal':
      return Neighborhood.PANTANAL;
    case 'trindade':
      return Neighborhood.TRINDADE;
    case 'carvoeira':
      return Neighborhood.CARVOEIRA;
    case 'saco dos limões':
      return Neighborhood.SACO_DOS_LIMOES;
    default:
      return Neighborhood.CORREGO_GRANDE;
  }
}

String getNeighborgoodString(Neighborhood neighborhood) {
  switch (neighborhood) {
    case Neighborhood.PANTANAL:
      return 'Pantanal';
    case Neighborhood.TRINDADE:
```

```
      return 'Trindade';
    case Neighborhood.CARVOEIRA:
      return 'Carvoeira';
    case Neighborhood.SACO_DOS_LIMOES:
      return 'Saco dos Limões';
    default:
      return 'Corrégo Grande';
  }
}
```

::::::::::::::

core/models/destination_model.dart

::::::::::::::

```
import 'package:google_maps_flutter/google_maps_flutter.dart';

class Destination {
  Destination({
    required this.position,
    required this.name,
    required this.distance,
    required this.duration,
  });

  Destination.fromMap(Map<String, dynamic> map)
      : name = map['destination']['name'] as String,
        distance = map['distance']['value'] as int,
        duration = Duration(seconds: map['duration']['value'] as int),
        position = LatLng.fromJson(
            _getCoordinates(map['destination']['coordinates'] as String))!;

  LatLng position;
  String name;
  int distance; // in meters
  Duration duration;
}

List<double> _getCoordinates(String coordinates) {
  final List<String> strings = coordinates.split(', ');
```

```
  return strings.map((String e) => double.parse(e)).toList();
}
::::::::::::::
core/models/listing_model.dart
::::::::::::::
import 'package:mobile/core/models/destination_model.dart';
import 'package:mobile/core/models/trip_model.dart';

import 'address_model.dart';

class Listing {
  Listing({
    required this.id,
    required this.priceMonthly,
    required this.condoFeeMonthly,
    required this.propertyTaxMonthly,
    required this.squareMeters,
    required this.title,
    required this.description,
    required this.phoneNumbers,
    required this.picturesUrls,
    required this.address,
    required this.distances,
    required this.amenities,
    required this.parkingSpaces,
    required this.bedrooms,
    required this.unitType,
  });

  Listing.fromDocumentSnapshot(Map<String, dynamic> document)
      : id = document['id'] as String,
        priceMonthly = int.parse(document['price'] as String),
        condoFeeMonthly = document.containsKey('condoFee')
            ? int.parse(document['condoFee'] as String)
            : 0,
        propertyTaxMonthly = document.containsKey('iptu')
            ? int.parse(document['iptu'] as String)
```

```
        : 0,
    squareMeters = int.parse(document['usableArea'] as String),
    title = document['title'] as String,
    description = document['description'] as String,
    phoneNumbers = document['phonenumbers'] as List<dynamic>,
    picturesUrls = document['picturesUri'] as List<dynamic>,
    address = Address.fromMap(document['address'] as Map<String, dynamic>),
    distances = (document['distances'] as List<dynamic>)
        .map<Trip>(
            (dynamic trip) => Trip.fromMap(trip as Map<String, dynamic>))
        .toList(),
    amenities = document['amenities'] as List<dynamic>,
    parkingSpaces = document.containsKey('parkingSpaces')
        ? document['parkingSpaces'] as int
        : 0,
    bedrooms =
        document.containsKey('bedrooms') ? document['bedrooms'] as int : 0,
    unitType = _getUnitType(document['unitType'] as String);

String id;
int priceMonthly;
int condoFeeMonthly;
int propertyTaxMonthly;
int squareMeters;
String title;
String description;
List<dynamic> phoneNumbers;
List<dynamic> picturesUrls;
Address address;
List<Trip> distances;
List<dynamic> amenities;
int parkingSpaces;
String _department = '';
int bedrooms;
UnitType unitType;

set department(String department) {
```

```
  _department = department;
}

Duration get distanceWalking {
  if (_department.isEmpty) {
    return Duration.zero; //FIXME
  }

  final Destination targetDestination = distances
      .firstWhere((Trip element) => element.mode == TripMode.WALKING)
      .destinations
      .firstWhere((Destination element) => element.name == _department);

  return targetDestination.duration;
}

Duration get distanceBicycle {
  if (_department.isEmpty) {
    return Duration.zero; //FIXME
  }

  final Destination targetDestination = distances
      .firstWhere((Trip element) => element.mode == TripMode.BICYCLE)
      .destinations
      .firstWhere((Destination element) => element.name == _department);

  return targetDestination.duration;
}

bool get hasLaundry => amenities.contains('LAUNDRY');
bool get isFurnished => amenities.contains('FURNISHED');
bool get hasPool => amenities.contains('POOL');
bool get hasGrill => amenities.contains('BARBECUE_GRILL');
bool get hasElevator => amenities.contains('ELEVATOR');
bool get hasGym => amenities.contains('GYM');
bool get hasAirConditioning => amenities.contains('AIR CONDITIONING');
bool get petsAllowed => amenities.contains('PETS_ALLOWED');
```

```
  bool get hasParking => parkingSpaces > 0;

  int get totalPrice => priceMonthly + condoFeeMonthly + propertyTaxMonthly;
}

enum UnitType {
  KITNET,
  APARTMENT,
  HOUSE,
}

UnitType _getUnitType(String value) {
  switch (value) {
    case 'KITNET':
      return UnitType.KITNET;
    case 'HOUSE':
      return UnitType.HOUSE;
    default:
      return UnitType.APARTMENT;
  }
}

String getStringFromUnitType(UnitType value) {
  switch (value) {
    case UnitType.KITNET:
      return 'KITNET';
    case UnitType.HOUSE:
      return 'HOUSE';
    default:
      return 'APARTMENT';
  }
}
```

::::::::::::::

core/models/trip_model.dart

::::::::::::::

```
import 'package:mobile/core/models/destination_model.dart';
```

```
enum TripMode {
  WALKING,
  BICYCLE,
}

class Trip {
  Trip({required this.mode, required this.destinations});

  Trip.fromMap(Map<String, dynamic> map)
      : mode = _getEnum(map['mode'] as String),
        destinations = (map['trips'] as List<dynamic>)
            .map<Destination>((dynamic destination) =>
                Destination.fromMap(destination as Map<String, dynamic>))
            .toList();

  TripMode mode;
  List<Destination> destinations;
}

TripMode _getEnum(String value) {
  if (value == 'walking') {
    return TripMode.WALKING;
  }

  return TripMode.BICYCLE;
}
```

```
::::::::::::::
core/tenant/departments/department_provider.dart
::::::::::::::
```

```
import 'package:flutter/widgets.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:shared_preferences/shared_preferences.dart';

class DepartmentProvider extends ChangeNotifier {
  DepartmentProvider() {
    SharedPreferences.getInstance().then((SharedPreferences value) {
      _prefs = value;
```

```
    _selected = _getFromStorage();
    notifyListeners();
  });
}

static const String _SHARED_PREFERENCES_KEY = 'DEPARTMENT';
late SharedPreferences _prefs;

final Map<String, LatLng> _departments = <String, LatLng>{
  'CTC - UFSC': const LatLng(-27.600303621950385, -48.51779077562106),
  'CCA - UFSC': const LatLng(-27.582073584735593, -48.504128482231366),
  'CCB - UFSC': const LatLng(-27.598426298259692, -48.51451541499237),
  'CCE - UFSC': const LatLng(-27.600929360715952, -48.52176000778667),
  'CCS - UFSC': const LatLng(-27.598944540256333, -48.517589929667295),
  'CCJ - UFSC': const LatLng(-27.598302330575038, -48.52208040276732),
  'CDS - UFSC': const LatLng(-27.603081351068045, -48.51940576825233),
  'CED - UFSC': const LatLng(-27.6018397831294, -48.522324463350905),
  'CFH - UFSC': const LatLng(-27.601935474427318, -48.523022312427756),
  'CFM - UFSC': const LatLng(-27.601433121602625, -48.52232681120341),
  'CSE - UFSC': const LatLng(-27.599482651151753, -48.52142025490006),
};
String? _selected;

List<String> get departments => _departments.keys.toList();

String? get selected => _selected;
LatLng get position => _departments[_selected]!;

set selected(String? department) {
  _selected = department;
  _persistInStorage();
  notifyListeners();
}

void _persistInStorage() {
  if (_selected != null) {
    _prefs.setString(_SHARED_PREFERENCES_KEY, _selected!);
```

```
    }
  }

  String? _getFromStorage() => _prefs.getString(_SHARED_PREFERENCES_KEY);
}
::::::::::::::
core/tenant/favorites/favorites_provider.dart
::::::::::::::
import 'package:flutter/widgets.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:shared_preferences/shared_preferences.dart';

class FavoritesProvider extends ChangeNotifier {
  FavoritesProvider() {
    SharedPreferences.getInstance().then((SharedPreferences value) {
      _prefs = value;
      _ids = _getFromStorage();
      notifyListeners();
    });
  }

  static const String _SHARED_PREFERENCES_KEY = 'FAVORITES';
  late SharedPreferences _prefs;
  Set<String> _ids = <String>{};
  List<String> get favorites => _ids.toList();

  bool add(Listing listing) {
    final bool wasAlreadyInFavorites = _ids.add(listing.id);

    _persistInStorage();
    notifyListeners();

    return wasAlreadyInFavorites;
  }

  void remove(Listing listing) {
    _ids.removeWhere((String id) => listing.id == id);
```

```
    _persistInStorage();
    notifyListeners();
  }

  bool contains(Listing listing) => _ids.contains(listing.id);

  void _persistInStorage() {
    _prefs.setStringList(_SHARED_PREFERENCES_KEY, _ids.toList());
  }

  Set<String> _getFromStorage() {
    final List<String> list =
        _prefs.getStringList(_SHARED_PREFERENCES_KEY) ?? <String>[];

    return Set<String>.from(list);
  }
}
::::::::::::::
core/tenant/favorites/filtered_favorites_provider.dart
::::::::::::::
import 'package:flutter/widgets.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/core/tenant/favorites/favorites_provider.dart';

class FilteredFavoritesProvider extends ChangeNotifier {
  FilteredFavoritesProvider();

  FilteredFavoritesProvider.update(
    List<Listing> allListings,
    FavoritesProvider favorites,
  ) {
    listings = List<Listing>.from(allListings);

    listings.retainWhere(favorites.contains);

    notifyListeners();
```

```
  }

  List<Listing> listings = List<Listing>.empty();
}
::::::::::::::
core/tenant/filters/filter_provider.dart
::::::::::::::
import 'dart:math';

import 'package:flutter/widgets.dart';
import 'package:mobile/core/models/address_model.dart';

enum SortBy {
  PRICE,
  WALKING,
  BICYCLE,
}

class FilterProvider extends ChangeNotifier {
  Point<int> _priceRange = const Point<int>(0, 10000);
  Duration _maxDistanceWalking = const Duration(minutes: 45);
  Duration _maxDistanceBicycle = const Duration(minutes: 30);
  int _minSquareMeters = 0;
  bool _hasLaundry = false;
  bool _isFurnished = false;
  bool _hasPool = false;
  bool _hasGrill = false;
  bool _hasElevator = false;
  bool _hasGym = false;
  bool _hasAirConditioning = false;
  bool _petsAllowed = false;
  bool _hasParking = false;
  int _minBedrooms = 0;
  bool _isApartment = false;
  bool _isKitnet = false;
  bool _isHouse = false;
  SortBy _sortBy = SortBy.WALKING;
```

```
Set<Neighborhood> _neighborhoods = <Neighborhood>{};

set priceRange(Point<int> priceRange) {
  _priceRange = priceRange;
  notifyListeners();
}

Point<int> get priceRange => _priceRange;

set maxDistanceWalking(Duration maxDistanceWalking) {
  _maxDistanceWalking = maxDistanceWalking;
  notifyListeners();
}

Duration get maxDistanceWalking => _maxDistanceWalking;

set maxDistanceBicycle(Duration maxDistanceBicycle) {
  _maxDistanceBicycle = maxDistanceBicycle;
  notifyListeners();
}

Duration get maxDistanceBicycle => _maxDistanceBicycle;

set minSquareMeters(int minSquareMeters) {
  _minSquareMeters = minSquareMeters;
  notifyListeners();
}

int get minSquareMeters => _minSquareMeters;

set hasLaundry(bool hasLaundry) {
  print(hasLaundry);
  _hasLaundry = hasLaundry;
  notifyListeners();
}

bool get hasLaundry => _hasLaundry;
```

```
 set isFurnished(bool isFurnished) {
  _isFurnished = isFurnished;
  notifyListeners();
 }

 bool get isFurnished => _isFurnished;

 set hasPool(bool hasPool) {
  _hasPool = hasPool;
  notifyListeners();
 }

 bool get hasPool => _hasPool;

 set hasGrill(bool hasGrill) {
  _hasGrill = hasGrill;
  notifyListeners();
 }

 bool get hasGrill => _hasGrill;

 set hasElevator(bool hasElevator) {
  _hasElevator = hasElevator;
  notifyListeners();
 }

 bool get hasElevator => _hasElevator;

 set hasGym(bool hasGym) {
  _hasGym = hasGym;
  notifyListeners();
 }

 bool get hasGym => _hasGym;

 set hasAirConditioning(bool hasAirConditioning) {
```

```
  _hasAirConditioning = hasAirConditioning;
  notifyListeners();
 }

 bool get hasAirConditioning => _hasAirConditioning;

 set petsAllowed(bool petsAllowed) {
  _petsAllowed = petsAllowed;
  notifyListeners();
 }

 bool get petsAllowed => _petsAllowed;

 set hasParking(bool hasParking) {
  _hasParking = hasParking;
  notifyListeners();
 }

 bool get hasParking => _hasParking;

 set minBedrooms(int minBedrooms) {
  _minBedrooms = minBedrooms;
  notifyListeners();
 }

 int get minBedrooms => _minBedrooms;

 set isApartment(bool isApartment) {
  _isApartment = isApartment;
  notifyListeners();
 }

 bool get isApartment => _isApartment;

 set isKitnet(bool isKitnet) {
  _isKitnet = isKitnet;
  notifyListeners();
```

```
  }

  bool get isKitnet => _isKitnet;

  set isHouse(bool isHouse) {
    _isHouse = isHouse;
    notifyListeners();
  }

  bool get isHouse => _isHouse;

  void setNeighborhoods(Neighborhood neighborhoods, bool insert) {
    if (insert) {
      _neighborhoods.add(neighborhoods);
    } else {
      _neighborhoods.remove(neighborhoods);
    }
    notifyListeners();
  }

  List<Neighborhood> get neighborhoods => _neighborhoods.toList();

  bool isNeighborhoodSelected(Neighborhood neighborhood) =>
      _neighborhoods.contains(neighborhood);

  void clear() {
    _priceRange = const Point<int>(0, 10000);
    _maxDistanceWalking = const Duration(minutes: 45);
    _maxDistanceBicycle = const Duration(minutes: 30);
    _minSquareMeters = 0;
    _hasLaundry = false;
    _isFurnished = false;
    _hasPool = false;
    _hasGrill = false;
    _hasElevator = false;
    _hasGym = false;
    _hasAirConditioning = false;
```

```
    _petsAllowed = false;
    _hasParking = false;
    _minBedrooms = 0;
    _isApartment = false;
    _isKitnet = false;
    _isHouse = false;
    _neighborhoods = <Neighborhood>{};

    notifyListeners();
  }

  set sortBy(SortBy sortBy) {
    _sortBy = sortBy;
    notifyListeners();
  }

  SortBy get sortBy => _sortBy;
}
::::::::::::::
core/tenant/listings/filtered_listings_provider.dart
::::::::::::::
import 'package:flutter/widgets.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/core/tenant/filters/filter_provider.dart';

class FilteredListingsProvider extends ChangeNotifier {
  FilteredListingsProvider();

  FilteredListingsProvider.update(
    List<Listing> allListings,
    FilterProvider this.filters,
    String? selectedDepartment,
  ) {
    if (selectedDepartment != null) {
      listings = List<Listing>.from(allListings).map((Listing listing) {
        listing.department = selectedDepartment;
        return listing;
```

```
  }).toList();
 } else {
  listings = List<Listing>.from(allListings);
 }

 listings.retainWhere(testPrice);
 listings.retainWhere(testWalking);
 listings.retainWhere(testBicycle);
 listings.retainWhere(testSquareMeters);
 listings.retainWhere(testBedrroms);
 if (filters!.hasLaundry) {
  listings.retainWhere(testLaundry);
 }
 if (filters!.isFurnished) {
  listings.retainWhere(testFurnished);
 }
 if (filters!.hasPool) {
  listings.retainWhere(testPool);
 }
 if (filters!.hasGrill) {
  listings.retainWhere(testGrill);
 }
 if (filters!.hasElevator) {
  listings.retainWhere(testElevator);
 }
 if (filters!.hasGym) {
  listings.retainWhere(testGym);
 }
 if (filters!.hasAirConditioning) {
  listings.retainWhere(testAc);
 }
 if (filters!.petsAllowed) {
  listings.retainWhere(testPets);
 }
 if (filters!.hasParking) {
  listings.retainWhere(testParking);
 }
```

```
  if (filters!.isApartment) {
    listings.retainWhere(testApartment);
  } else if (filters!.isKitnet) {
    listings.retainWhere(testKitnet);
  } else if (filters!.isHouse) {
    listings.retainWhere(testHouse);
  }
  if (filters!.neighborhoods.isNotEmpty) {
    listings.retainWhere((Listing listing) =>
        filters!.isNeighborhoodSelected(listing.address.neighborhood));
  }

  listings.sort((Listing a, Listing b) {
    switch (filters!.sortBy) {
      case SortBy.PRICE:
        return priceComparator(a, b);
      case SortBy.WALKING:
        return walkingComparator(a, b);
      case SortBy.BICYCLE:
        return bicycleComparator(a, b);
    }
  });

  notifyListeners();
}

 List<Listing> listings = List<Listing>.empty();
 FilterProvider? filters;

 bool testPrice(Listing listing) =>
    listing.totalPrice <= filters!.priceRange.y;

 bool testWalking(Listing listing) =>
    listing.distanceWalking <= filters!.maxDistanceWalking;

 bool testBicycle(Listing listing) =>
    listing.distanceBicycle <= filters!.maxDistanceBicycle;
```

```
bool testSquareMeters(Listing listing) =>
    listing.squareMeters >= filters!.minSquareMeters;

bool testLaundry(Listing listing) => listing.hasLaundry == true;

bool testFurnished(Listing listing) => listing.isFurnished == true;

bool testPool(Listing listing) => listing.hasPool == true;

bool testGrill(Listing listing) => listing.hasGrill == true;

bool testElevator(Listing listing) => listing.hasElevator == true;

bool testGym(Listing listing) => listing.hasGym == true;

bool testAc(Listing listing) => listing.hasAirConditioning == true;

bool testPets(Listing listing) => listing.petsAllowed == true;

bool testParking(Listing listing) => listing.hasParking == true;

// end amenities

bool testBedrroms(Listing listing) =>
    listing.bedrooms >= filters!.minBedrooms;

bool testApartment(Listing listing) => listing.unitType == UnitType.APARTMENT;

bool testKitnet(Listing listing) => listing.unitType == UnitType.KITNET;

bool testHouse(Listing listing) => listing.unitType == UnitType.HOUSE;

// comparators

int priceComparator(Listing a, Listing b) {
  if (a.totalPrice == b.totalPrice) {
```

```
      return 0;
    }
    if (a.totalPrice < b.totalPrice) {
      return -1;
    } else {
      return 1;
    }
  }

  int walkingComparator(Listing a, Listing b) {
    if (a.distanceWalking == b.distanceWalking) {
      return 0;
    }
    if (a.distanceWalking < b.distanceWalking) {
      return -1;
    } else {
      return 1;
    }
  }

  int bicycleComparator(Listing a, Listing b) {
    if (a.distanceBicycle == b.distanceBicycle) {
      return 0;
    }
    if (a.distanceBicycle < b.distanceBicycle) {
      return -1;
    } else {
      return 1;
    }
  }
}
::::::::::::::
core/tenant/listings/listings_provider.dart
::::::::::::::
import 'package:flutter/foundation.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/core/tenant/listings/listings_service.dart';
```

```
class ListingsProvider extends ChangeNotifier {
  ListingsProvider() {
    _service = ListingsService();

    _fetchListings();
  }

  late ListingsService _service;
  bool _isLoading = false;
  List<Listing> _listings = List<Listing>.empty();

  set isLoading(bool isLoading) {
    _isLoading = isLoading;
    notifyListeners();
  }

  bool get isLoading => _isLoading;

  set listings(List<Listing> listings) {
    _listings = listings;
    notifyListeners();
  }

  List<Listing> get listings => _listings;

  Future<void> _fetchListings() async {
    isLoading = true;
    listings = await _service.fetchAll();
    isLoading = false;
  }
}
```

::::::::::::::

core/tenant/listings/listings_service.dart

::::::::::::::

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:mobile/core/models/listing_model.dart';
```

```
class ListingsService {
  Future<List<Listing>> fetchAll() async {
    final FirebaseFirestore firestore = FirebaseFirestore.instance;

    final QuerySnapshot<Map<String, dynamic>> snapshot =
        await firestore.collection('listings').get();

    return snapshot.docs
        .map((QueryDocumentSnapshot<Map<String, dynamic>> document) =>
            Listing.fromDocumentSnapshot(document.data()))
        .toList();
  }
}
```

:::::::::::::

main.dart

:::::::::::::

```
import 'dart:async';

import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_crashlytics/firebase_crashlytics.dart';
import 'package:flutter/material.dart';
import 'package:flutterfire_ui/i10n.dart';
import 'package:mobile/core/landlord/listings/listings_provider.dart';
import 'package:mobile/core/tenant/departments/department_provider.dart';
import 'package:mobile/core/tenant/favorites/favorites_provider.dart';
import 'package:mobile/core/tenant/favorites/filtered_favorites_provider.dart';
import 'package:mobile/core/tenant/filters/filter_provider.dart';
import 'package:mobile/core/tenant/listings/filtered_listings_provider.dart';
import 'package:mobile/core/tenant/listings/listings_provider.dart';
import 'package:mobile/ui/i18n/label_overrides.dart';
import 'package:mobile/ui/screens/tenant/map/map_screen.dart';
import 'package:mobile/ui/screens/welcome/welcome_screen.dart';
import 'package:provider/provider.dart';

void main() {
  runZonedGuarded<Future<void>>(() async {
```

```
    WidgetsFlutterBinding.ensureInitialized();
    await Firebase.initializeApp();
    FlutterError.onError = FirebaseCrashlytics.instance.recordFlutterError;

    runApp(const MyApp());
  },
      (Object error, StackTrace stack) =>
          FirebaseCrashlytics.instance.recordError(error, stack));
}

class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) => MultiProvider(
        // ignore: always_specify_types
        providers: [
          // this probably can be lower in the tree after the onboarding screens
          ChangeNotifierProvider<ListingsProvider>(
            create: (_) => ListingsProvider(),
          ),
          ChangeNotifierProvider<LandlordListingsProvider>(
            create: (_) => LandlordListingsProvider(),
          ),
          // can this be lower in the tree?
          ChangeNotifierProvider<FilterProvider>(
            create: (_) => FilterProvider(),
          ),
          // can this be lower in the tree?
          ChangeNotifierProvider<FavoritesProvider>(
            create: (_) => FavoritesProvider(),
          ),
```

```
    ChangeNotifierProvider<DepartmentProvider>(
      create: (_) => DepartmentProvider(),
    ),
    ChangeNotifierProxyProvider3<ListingsProvider, FilterProvider,
        DepartmentProvider, FilteredListingsProvider>(
      create: (_) => FilteredListingsProvider(),
      update: (
        _,
        ListingsProvider listings,
        FilterProvider filters,
        DepartmentProvider department,
        __,
      ) =>
          FilteredListingsProvider.update(
        listings.listings,
        filters,
        department.selected,
      ),
      lazy: false,
    ),
    ChangeNotifierProxyProvider2<ListingsProvider, FavoritesProvider,
        FilteredFavoritesProvider>(
      create: (_) => FilteredFavoritesProvider(),
      update: (
        _,
        ListingsProvider listings,
        FavoritesProvider favorites,
        __,
      ) =>
          FilteredFavoritesProvider.update(
        listings.listings,
        favorites,
      ),
      lazy: false,
    ),
  ],
  child: MaterialApp(
```

```
        title: 'TCC UFSC',
        home: Consumer<DepartmentProvider>(
           builder: (_, DepartmentProvider provider, __) {
          final bool isDepartmentSelected = provider.selected != null;

          return isDepartmentSelected
              ? const MapScreen()
              : const WelcomeScreen();
        }),
        localizationsDelegates: <LocalizationsDelegate<dynamic>>[
          // Creates an instance of FirebaseUILocalizationDelegate with overridden labels
          FlutterFireUILocalizations.withDefaultOverrides(
              const LabelOverrides()),

          // Delegates below take care of built-in flutter widgets
          GlobalMaterialLocalizations.delegate,
          GlobalWidgetsLocalizations.delegate,

          // This delegate is required to provide the labels that are not overridden by
LabelOverrides
          FlutterFireUILocalizations.delegate,
        ],
      ),
    );
}
::::::::::::::
ui/common/listing/listing_item.dart
::::::::::::::
import 'package:flutter/material.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/ui/common/listing/listing_item_image_slider.dart';
import 'package:mobile/ui/common/listing/listing_item_info.dart';
import 'package:mobile/ui/screens/tenant/details/details_screen.dart';

class ListingItem extends StatelessWidget {
  const ListingItem(this.listing,
    {Key? key, this.hideDistances = false, this.onTap})
```

```
    : super(key: key);

final Listing listing;
final bool hideDistances;
final Function? onTap;

@override
Widget build(BuildContext context) => InkWell(
      onTap: () {
        if (onTap != null) {
          onTap!();
        } else {
          Navigator.of(context).push<dynamic>(MaterialPageRoute<dynamic>(
            builder: (_) => DetailsScreen(listing),
          ));
        }
      },
      child: Card(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            ListingItemImageSlider(listing.picturesUrls),
            Padding(
              padding:
                  const EdgeInsets.symmetric(horizontal: 16, vertical: 8),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: <Widget>[
                  Text(
                    listing.title,
                    style: Theme.of(context).textTheme.caption,
                  ),
                  const SizedBox(height: 6),
                  Text(
                    listing.address.street,
                    style: Theme.of(context).textTheme.subtitle2,
                  ),
```

```
                Text(
                  '${listing.address.neighborhoodString}, ${listing.address.city}',
                  style: Theme.of(context).textTheme.caption,
                ),
                const SizedBox(height: 6),
                ListingItemInfo(listing, hideDistances: hideDistances),
              ],
            ),
          ),
        ],
      ),
    ),
  );
}
```

::::::::::::::

ui/common/listing/listing_item_image_slider.dart

::::::::::::::

```
import 'package:cached_network_image/cached_network_image.dart';
import 'package:carousel_slider/carousel_slider.dart';
import 'package:flutter/material.dart';

class ListingItemImageSlider extends StatelessWidget {
  const ListingItemImageSlider(this.picturesUrls, {Key? key}) : super(key: key);

  final List<dynamic> picturesUrls;

  Widget buildItems(BuildContext context, int index, int pageViewIndex) {
    final String url = picturesUrls[index] as String;

    return SizedBox(
      width: double.infinity,
      child: Stack(
        fit: StackFit.expand,
        children: <Widget>[
          CachedNetworkImage(
            imageUrl: url,
            key: Key(url),
```

```
          fit: BoxFit.fitWidth,
          placeholder: (_, __) => const Center(
            child: SizedBox(
              height: 50,
              width: 50,
              child: CircularProgressIndicator(),
            ),
          ),
          errorWidget: (_, __, dynamic ___) => const Icon(Icons.error),
        ),
        Align(
          alignment: Alignment.bottomCenter,
          child: Padding(
            padding: const EdgeInsets.only(bottom: 8),
            child: Row(
              mainAxisSize: MainAxisSize.min,
              children: getIndicators(index, picturesUrls.length),
            ),
          ),
        )
      ],
    ),
  );
}

 List<Widget> getIndicators(int index, int count) {
  final List<Widget> list = List<Widget>.empty(growable: true);

  for (int i = 0; i < count; i++) {
    list.add(Icon(
      index == i ? Icons.circle : Icons.radio_button_unchecked,
      color: Colors.white,
      size: 12,
    ));
  }

  return list;
```

```
  }

  @override
  Widget build(BuildContext context) {
    return CarouselSlider.builder(
      options: CarouselOptions(
        enableInfiniteScroll: false,
        viewportFraction: 1.0,
        enlargeCenterPage: false,
      ),
      itemCount: picturesUrls.length,
      itemBuilder: buildItems,
    );
  }
}
::::::::::::::
ui/common/listing/listing_item_info.dart
::::::::::::::
import 'package:flutter/material.dart';
import 'package:mobile/core/models/listing_model.dart';

class ListingItemInfo extends StatelessWidget {
  const ListingItemInfo(
    this.listing, {
    Key? key,
    this.hideDistances = false,
  }) : super(key: key);

  final Listing listing;
  final bool hideDistances;

  @override
  Widget build(BuildContext context) {
    return Wrap(
      spacing: 8,
      children: <Widget>[
        Chip(
```

```
    visualDensity: VisualDensity.compact,
    avatar: const Icon(Icons.attach_money),
    label: Text(listing.totalPrice.toString()),
  ),
  if (!hideDistances)
    Chip(
      visualDensity: VisualDensity.compact,
      avatar: const Icon(Icons.directions_bike),
      label: Text('${listing.distanceBicycle.inMinutes.toString()} min'),
    ),
  if (!hideDistances)
    Chip(
      visualDensity: VisualDensity.compact,
      avatar: const Icon(Icons.directions_walk),
      label: Text('${listing.distanceWalking.inMinutes.toString()} min'),
    ),
  if (listing.bedrooms > 0)
    Chip(
      visualDensity: VisualDensity.compact,
      avatar: const Icon(Icons.bed),
      label: Text('${listing.bedrooms} Quartos'),
    ),
  if (listing.hasAirConditioning)
    const Chip(
      visualDensity: VisualDensity.compact,
      avatar: Icon(Icons.ac_unit),
      label: Text('AC'),
    ),
  if (listing.hasParking)
    const Chip(
      visualDensity: VisualDensity.compact,
      avatar: Icon(Icons.local_parking),
      label: Text('Garagem'),
    ),
  if (listing.isFurnished)
    const Chip(
      visualDensity: VisualDensity.compact,
```

```
      avatar: Icon(Icons.bed),
      label: Text('Mobiliado'),
    ),
  if (listing.hasLaundry)
    const Chip(
      visualDensity: VisualDensity.compact,
      avatar: Icon(Icons.local_laundry_service),
      label: Text('Lavanderia'),
    ),
  if (listing.hasPool)
    const Chip(
      visualDensity: VisualDensity.compact,
      avatar: Icon(Icons.pool),
      label: Text('Piscina'),
    ),
  if (listing.hasGrill)
    const Chip(
      visualDensity: VisualDensity.compact,
      avatar: Icon(Icons.outdoor_grill),
      label: Text('Churrasqueira'),
    ),
  if (listing.hasElevator)
    const Chip(
      visualDensity: VisualDensity.compact,
      avatar: Icon(Icons.elevator),
      label: Text('Elevador'),
    ),
  if (listing.hasGym)
    const Chip(
      visualDensity: VisualDensity.compact,
      avatar: Icon(Icons.fitness_center),
      label: Text('Academia'),
    ),
  if (listing.petsAllowed)
    const Chip(
      visualDensity: VisualDensity.compact,
      avatar: Icon(Icons.pets),
```

```
            label: Text('Pets'),
          ),
      ],
    );
  }
}
::::::::::::::
ui/i18n/label_overrides.dart
::::::::::::::
import 'package:flutterfire_ui/i10n.dart';

class LabelOverrides extends DefaultLocalizations {
  const LabelOverrides();

  @override
  String get signInWithGoogleButtonText => 'Entrar usando Google';
}
::::::::::::::
ui/screens/landlord/auth_gate/auth_gate_screen.dart
::::::::::::::
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:flutterfire_ui/auth.dart';
import 'package:mobile/ui/screens/landlord/list/list_screen.dart';

class AuthGateScreen extends StatefulWidget {
  const AuthGateScreen({Key? key}) : super(key: key);

  @override
  State<AuthGateScreen> createState() => _AuthGateScreenState();
}

class _AuthGateScreenState extends State<AuthGateScreen> {
  @override
  void initState() {
    super.initState();
```

```
    FirebaseAuth.instance.authStateChanges().listen((User? event) {
      if (event != null) {
        Future<dynamic>.microtask(
          () => Navigator.push(
            context,
            MaterialPageRoute<Widget>(
              builder: (_) => const LandlordListScreen(),
            ),
          ),
        );
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(),
        body: const SignInScreen(
          providerConfigs: <ProviderConfiguration>[
            EmailProviderConfiguration(),
            GoogleProviderConfiguration(
              clientId:

'293946693616-706l7bqi6dqg9bcjlv8ps1v6q7hbd3us.apps.googleusercontent.com',
            ),
          ],
        ));
  }
}
::::::::::::::
ui/screens/landlord/common/common_bottom_navigation.dart
::::::::::::::
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:mobile/ui/screens/welcome/welcome_screen.dart';
```

```
enum BottomNavigationIndex {
  LIST,
  LOGOUT,
}

class CommonBottomNavigation extends StatelessWidget {
  const CommonBottomNavigation({Key? key, required this.currentIndex})
      : super(key: key);

  final BottomNavigationIndex currentIndex;

  void _onTap(int index, BuildContext context) {
    late Widget? nextPage;

    switch (BottomNavigationIndex.values[index]) {
      case BottomNavigationIndex.LIST:
        nextPage = null;
        break;
      case BottomNavigationIndex.LOGOUT:
        FirebaseAuth.instance.signOut();
        nextPage = const WelcomeScreen();

        break;
    }

    if (nextPage != null) {
      // Push next page without animation
      Navigator.of(context).push<dynamic>(PageRouteBuilder<dynamic>(
        pageBuilder: (_, __, ___) => nextPage!,
        transitionDuration: Duration.zero,
      ));
    }
  }

  @override
  Widget build(BuildContext context) {
    return BottomNavigationBar(
```

```
      currentIndex: currentIndex.index,
      items: <BottomNavigationBarItem>[
        const BottomNavigationBarItem(
          icon: Icon(Icons.list),
          label: 'Lista',
        ),
        const BottomNavigationBarItem(
          icon: Icon(Icons.logout),
          label: 'Deslogar',
        ),
      ],
      onTap: (int index) => _onTap(index, context),
    );
  }
}
::::::::::::::
ui/screens/landlord/details/details_body.dart
::::::::::::::
import 'package:flutter/material.dart';
import 'package:flutter_widget_from_html_core/flutter_widget_from_html_core.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/ui/common/listing/listing_item_image_slider.dart';
import 'package:mobile/ui/common/listing/listing_item_info.dart';
import 'package:mobile/ui/screens/tenant/details/details_prices.dart';
import 'package:url_launcher/url_launcher.dart';

class DetailsBody extends StatelessWidget {
  const DetailsBody(this.listing, {Key? key}) : super(key: key);

  final Listing listing;

  List<Widget> getContacts() {
    return listing.phoneNumbers.map((dynamic phoneNumber) {
      final String phone = phoneNumber as String;

      return TextButton.icon(
        icon: const Icon(Icons.phone),
```

```
      label: Text(phone),
      onPressed: () async {
        final String url = 'tel:$phone';
        if (await canLaunch(url)) {
          await launch(url);
        } else {
          throw 'Could not launch $url';
        }
      },
    );
  }).toList();
}

@override
Widget build(BuildContext context) => SingleChildScrollView(
      child: Column(
        children: <Widget>[
          ListingItemImageSlider(listing.picturesUrls),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: <Widget>[
                Text(
                  'Descrição',
                  style: Theme.of(context).textTheme.headline6,
                ),
                HtmlWidget(listing.description),
                const SizedBox(height: 8),
                Text(
                  'Valores',
                  style: Theme.of(context).textTheme.headline6,
                ),
                DetailsPrices(listing),
                const SizedBox(height: 8),
                Text(
                  'Características',
```

```
                style: Theme.of(context).textTheme.headline6,
              ),
              ListingItemInfo(
                listing,
                hideDistances: true,
              ),
              const SizedBox(height: 8),
              Text(
                'Contatos',
                style: Theme.of(context).textTheme.headline6,
              ),
              Column(
                children: getContacts(),
              ),
            ],
          ),
        )
      ],
    ),
  );
}
```

::::::::::::::

ui/screens/landlord/details/details_prices.dart

::::::::::::::

```
import 'package:flutter/material.dart';
import 'package:mobile/core/models/listing_model.dart';

class DetailsPrices extends StatelessWidget {
  const DetailsPrices(this.listing, {Key? key}) : super(key: key);

  final Listing listing;

  @override
  Widget build(BuildContext context) => Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: <Column>[
        Column(
```

```
            crossAxisAlignment: CrossAxisAlignment.start,
            children: <Widget>[
              const Text('Aluguel: '),
              const Text('Condomínio: '),
              const Text('IPTU (mensal): '),
              Text(
                'Valor total: ',
                style: Theme.of(context).textTheme.bodyText1,
              ),
            ],
          ),
          Column(
            crossAxisAlignment: CrossAxisAlignment.end,
            children: <Widget>[
              Text('R\$ ${listing.priceMonthly}'),
              Text('R\$ ${listing.condoFeeMonthly}'),
              Text('R\$ ${listing.propertyTaxMonthly}'),
              Text(
                'R\$ ${listing.totalPrice.toString()}',
                style: Theme.of(context).textTheme.bodyText1,
              ),
            ],
          ),
        ],
      );
}
```

::::::::::::::

ui/screens/landlord/details/details_screen.dart

::::::::::::::

```
import 'package:flutter/material.dart';
import 'package:mobile/core/landlord/listings/listings_provider.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/ui/screens/landlord/common/common_bottom_navigation.dart';
import 'package:mobile/ui/screens/landlord/details/details_body.dart';
import 'package:mobile/ui/screens/landlord/form/listing_form.dart';
import 'package:provider/provider.dart';
```

```
class LandlordDetailsScreen extends StatefulWidget {
  const LandlordDetailsScreen(this.listing, {Key? key}) : super(key: key);

  final Listing listing;

  @override
  State<LandlordDetailsScreen> createState() => _LandlordDetailsScreenState();
}

class _LandlordDetailsScreenState extends State<LandlordDetailsScreen> {
  late LandlordListingsProvider provider;

  @override
  void initState() {
    super.initState();
    provider = Provider.of<LandlordListingsProvider>(context, listen: false);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.listing.title),
        actions: <IconButton>[
          IconButton(
            onPressed: () {
              Navigator.of(context).push<dynamic>(MaterialPageRoute<dynamic>(
                builder: (_) => LandlordListingForm(
                  listing: widget.listing,
                ),
              ));
            },
            icon: const Icon(Icons.edit),
          ),
          IconButton(
            onPressed: () {
              provider.delete(widget.listing);
```

```
            Navigator.of(context).pop();
          },
          icon: const Icon(Icons.delete),
        ),
      ],
    ),
    body: DetailsBody(widget.listing),
    bottomNavigationBar: const CommonBottomNavigation(
      currentIndex: BottomNavigationIndex.LIST,
    ),
  );
 }
}
::::::::::::::
ui/screens/landlord/form/listing_form.dart
::::::::::::::
import 'dart:io';

import 'package:file_picker/file_picker.dart';
import 'package:flutter/material.dart';
import 'package:mobile/core/landlord/listings/listing_form_provider.dart';
import 'package:mobile/core/landlord/listings/listings_provider.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:provider/provider.dart';

class LandlordListingForm extends StatefulWidget {
 const LandlordListingForm({Key? key, this.listing}) : super(key: key);

 final Listing? listing;

 @override
 State<LandlordListingForm> createState() => _LandlordListingFormState();
}

class _LandlordListingFormState extends State<LandlordListingForm> {
 final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Cadastro de imóvel'),
    ),
    body: SingleChildScrollView(
      child: ChangeNotifierProvider<LandlordListingFormProvider>(
        create: (_) => LandlordListingFormProvider(
          context.read<LandlordListingsProvider>(),
          listing: widget.listing,
        ),
        child: Consumer<LandlordListingFormProvider>(
          builder: (_, LandlordListingFormProvider form, __) => Form(
            key: _formKey,
            child: Padding(
              padding: const EdgeInsets.all(16.0),
              child: Column(
                children: <Widget>[
                  TextFormField(
                    controller: TextEditingController(text: form.title),
                    decoration: const InputDecoration(
                      border: UnderlineInputBorder(),
                      labelText: 'Título',
                    ),
                    validator: (String? value) {
                      if (value == null || value.isEmpty) {
                        return 'O título não pode ser vazio!';
                      }
                      return null;
                    },
                    onSaved: (String? value) {
                      form.title = value;
                    },
                  ),
                  TextFormField(
                    minLines: 5,
```

```
  maxLines: 10,
  controller: TextEditingController(text: form.description),
  decoration: const InputDecoration(
    border: UnderlineInputBorder(),
    labelText: 'Descrição',
  ),
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'A descrição não pode ser vazia!';
    }
    return null;
  },
  onSaved: (String? value) {
    form.description = value;
  },
),
const SizedBox(height: 16),
Row(
  children: <Widget>[
    const Text('Imagens:'),
  ],
),
Column(
  children: form.picturesUris.map((File file) {
    final String filename =
        file.path.substring(file.path.lastIndexOf('/') + 1);
    return Text(filename);
  }).toList(),
),
FractionallySizedBox(
  widthFactor: 1,
  child: ElevatedButton(
    child: const Text('Adicionar imagens'),
    onPressed: () async {
      final FilePickerResult? result =
          await FilePicker.platform.pickFiles(
        type: FileType.image,
```

```
        allowMultiple: true,
      );

      if (result != null) {
        final List<File> files = result.paths
            .map((String? path) => File(path!))
            .toList();
        form.picturesUris = files;
      } else {
        // User canceled the picker
      }
    },
  ),
),
TextFormField(
  controller:
      TextEditingController(text: form.neighborhood),
  decoration: const InputDecoration(
    border: UnderlineInputBorder(),
    labelText: 'Bairro',
  ),
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'O bairro não pode ser vazio!';
    }
    return null;
  },
  onSaved: (String? value) {
    form.neighborhood = value;
  },
),
TextFormField(
  controller: TextEditingController(text: form.address),
  decoration: const InputDecoration(
    border: UnderlineInputBorder(),
    labelText: 'Endereço',
  ),
```

```
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'O endereço não pode ser vazio!';
    }
    return null;
  },
  onSaved: (String? value) {
    form.address = value;
  },
),
TextFormField(
  controller:
      TextEditingController(text: form.streetNumber),
  decoration: const InputDecoration(
    border: UnderlineInputBorder(),
    labelText: 'Número',
  ),
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'O número não pode ser vazio!';
    }
    return null;
  },
  onSaved: (String? value) {
    form.streetNumber = value;
  },
),
TextFormField(
  controller: TextEditingController(text: form.zipCode),
  decoration: const InputDecoration(
    border: UnderlineInputBorder(),
    labelText: 'CEP',
  ),
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'O CEP não pode ser vazio!';
    }
```

```
      return null;
    },
    onSaved: (String? value) {
      form.zipCode = value;
    },
  ),
  TextFormField(
    controller: TextEditingController(
        text: form.priceMonthly.toString()),
    keyboardType: TextInputType.number,
    decoration: const InputDecoration(
      border: UnderlineInputBorder(),
      labelText: 'Preço',
      suffixText: r'R$',
    ),
    validator: (String? value) {
      if (value == null || value.isEmpty) {
        return 'Preço precisa ser maior que 0!';
      }
      final int? number = int.tryParse(value);
      if (number == null || number <= 0) {
        return 'Preço precisa ser maior que 0!';
      }
      return null;
    },
    onSaved: (String? value) {
      form.priceMonthly = int.parse(value!);
    },
  ),
  TextFormField(
    controller: TextEditingController(
        text: form.condoFeeMonthly.toString()),
    keyboardType: TextInputType.number,
    decoration: const InputDecoration(
      border: UnderlineInputBorder(),
      labelText: 'Condomínio',
      suffixText: r'R$',
```

```
  ),
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'Preço do condomínio precisa ser maior que 0!';
    }
    final int? number = int.tryParse(value);
    if (number == null || number <= 0) {
      return 'Preço do condomínio precisa ser maior que 0!';
    }
    return null;
  },
  onSaved: (String? value) {
    form.condoFeeMonthly = int.parse(value!);
  },
),
TextFormField(
  controller: TextEditingController(
      text: form.propertyTaxMonthly.toString()),
  keyboardType: TextInputType.number,
  decoration: const InputDecoration(
    border: UnderlineInputBorder(),
    labelText: 'IPTU (mensal)',
    suffixText: r'R$',
  ),
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'Preço do IPTU precisa ser maior que 0!';
    }
    final int? number = int.tryParse(value);
    if (number == null || number <= 0) {
      return 'Preço do IPTU precisa ser maior que 0!';
    }
    return null;
  },
  onSaved: (String? value) {
    form.propertyTaxMonthly = int.parse(value!);
  },
```

```
),
TextFormField(
  controller:
      TextEditingController(text: form.bedrooms.toString()),
  keyboardType: TextInputType.number,
  decoration: const InputDecoration(
    border: UnderlineInputBorder(),
    labelText: 'Número de quartos',
  ),
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'Número de quartos precisa ser maior que 0!';
    }
    final int? number = int.tryParse(value);
    if (number == null || number <= 0) {
      return 'Número de quartos precisa ser maior que 0!';
    }
    return null;
  },
  onSaved: (String? value) {
    form.bedrooms = int.parse(value!);
  },
),
TextFormField(
  controller: TextEditingController(
      text: form.squareMeters.toString()),
  keyboardType: TextInputType.number,
  decoration: const InputDecoration(
    border: UnderlineInputBorder(),
    labelText: 'Área',
    suffixText: 'm2',
  ),
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'Área precisa ser maior que 0!';
    }
    final int? number = int.tryParse(value);
```

```
    if (number == null || number <= 0) {
      return 'Área precisa ser maior que 0!';
    }
    return null;
  },
  onSaved: (String? value) {
    form.squareMeters = int.parse(value!);
  },
),
TextFormField(
  controller: TextEditingController(
      text: form.parkingSpaces.toString()),
  keyboardType: TextInputType.number,
  decoration: const InputDecoration(
    border: UnderlineInputBorder(),
    labelText: 'Vagas na garagem',
  ),
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'Vagas na garem não pode ser vazio!';
    }
    final int? number = int.tryParse(value);
    if (number == null || number < 0) {
      return 'Preço do IPTU precisa ser maior ou igual a 0!';
    }
    return null;
  },
  onSaved: (String? value) {
    form.parkingSpaces = int.parse(value!);
  },
),
DropdownButtonFormField<UnitType>(
  value: form.unitType,
  decoration:
      const InputDecoration(labelText: 'Tipo de imovel'),
  onChanged: (UnitType? value) {
    form.unitType = value;
```

```
  },
  items: UnitType.values
      .map<DropdownMenuItem<UnitType>>((UnitType value) {
    return DropdownMenuItem<UnitType>(
      value: value,
      child: Text(value.toString()),
    );
  }).toList(),
),
TextFormField(
  controller:
      TextEditingController(text: form.phoneNumbers),
  decoration: const InputDecoration(
    border: UnderlineInputBorder(),
    labelText: 'Telefone de contato',
  ),
  validator: (String? value) {
    if (value == null || value.isEmpty) {
      return 'O contato não pode ser vazio!';
    }
    return null;
  },
  onSaved: (String? value) {
    form.phoneNumbers = value;
  },
),
const SizedBox(height: 16),
Row(
  children: <Widget>[
    const Text('Comodidades:'),
  ],
),
FractionallySizedBox(
  widthFactor: 1,
  child: Wrap(
    spacing: 8,
    children: <Widget>[
```

```
ChoiceChip(
  label: const Text('Ar condicionado'),
  avatar: const Icon(Icons.ac_unit),
  selected: form.hasAirConditioning,
  onSelected: (bool selected) {
    form.hasAirConditioning = selected;
  },
),
ChoiceChip(
  label: const Text('Mobiliado'),
  avatar: const Icon(Icons.bed),
  selected: form.isFurnished,
  onSelected: (bool selected) {
    form.isFurnished = selected;
  },
),
ChoiceChip(
  label: const Text('Lavanderia'),
  selected: form.hasLaundry,
  avatar: const Icon(Icons.local_laundry_service),
  onSelected: (bool selected) {
    form.hasLaundry = selected;
  },
),
ChoiceChip(
  label: const Text('Piscina'),
  avatar: const Icon(Icons.pool),
  selected: form.hasPool,
  onSelected: (bool selected) {
    form.hasPool = selected;
  },
),
ChoiceChip(
  label: const Text('Churrasqueira'),
  avatar: const Icon(Icons.outdoor_grill),
  selected: form.hasGrill,
  onSelected: (bool selected) {
```

```
            form.hasGrill = selected;
          },
        ),
        ChoiceChip(
          label: const Text('Elevador'),
          avatar: const Icon(Icons.elevator),
          selected: form.hasElevator,
          onSelected: (bool selected) {
            form.hasElevator = selected;
          },
        ),
        ChoiceChip(
          label: const Text('Academia'),
          avatar: const Icon(Icons.fitness_center),
          selected: form.hasGym,
          onSelected: (bool selected) {
            form.hasGym = selected;
          },
        ),
        ChoiceChip(
          label: const Text('Pets'),
          avatar: const Icon(Icons.pets),
          selected: form.petsAllowed,
          onSelected: (bool selected) {
            form.petsAllowed = selected;
          },
        ),
      ],
    ),
  ),
),
const SizedBox(height: 16),
FractionallySizedBox(
  widthFactor: 1,
  child: ElevatedButton(
      onPressed: () {
        if (_formKey.currentState!.validate()) {
          _formKey.currentState!.save();
```

```
                        form.send();
                        Navigator.of(context).pop();
                        Navigator.of(context).pop();
                      }
                    },
                    child: const Text('Enviar')),
                )
              ],
            ),
          ),
        ),
      ),
    ),
  );
 }
}
```

::::::::::::::

ui/screens/landlord/list/list_body.dart

::::::::::::::

```
import 'package:flutter/material.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/ui/common/listing/listing_item.dart';
import 'package:mobile/ui/screens/landlord/details/details_screen.dart';

class LandlordListBody extends StatelessWidget {
  const LandlordListBody({Key? key, required this.listings}) : super(key: key);

  final List<Listing> listings;

  @override
  Widget build(BuildContext context) {
    if (listings.isEmpty) {
      return const Center(child: Text('Nenhum anúncio encontrado'));
    }

    return ListView.builder(
```

```
      itemCount: listings.length,
      itemBuilder: (BuildContext context, int index) => ListingItem(
        listings[index],
        hideDistances: true,
        onTap: () {
          Navigator.of(context).push<dynamic>(MaterialPageRoute<dynamic>(
            builder: (_) => LandlordDetailsScreen(listings[index]),
          ));
        },
      ),
    );
  }
}
::::::::::::::
ui/screens/landlord/list/list_screen.dart
::::::::::::::
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:mobile/core/landlord/listings/listings_provider.dart';
import 'package:mobile/ui/screens/landlord/form/listing_form.dart';
import 'package:mobile/ui/screens/landlord/list/list_body.dart';
import 'package:mobile/ui/screens/welcome/welcome_screen.dart';
import 'package:provider/provider.dart';

class LandlordListScreen extends StatelessWidget {
  const LandlordListScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        title: const Text('Lista'),
      ),
      body: Consumer<LandlordListingsProvider>(
        builder: (_, LandlordListingsProvider provider, __) {
          return LandlordListBody(
```

```
          listings: provider.listings,
          key: Key(
            provider.listings.length.toString(), // FIXME
          ),
        );
      },
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () {
        Navigator.of(context).push<dynamic>(MaterialPageRoute<dynamic>(
          builder: (_) => const LandlordListingForm(),
        ));
      },
      child: const Icon(Icons.add),
    ),
    bottomNavigationBar: BottomNavigationBar(
      currentIndex: 0,
      items: <BottomNavigationBarItem>[
        const BottomNavigationBarItem(
          icon: Icon(Icons.list),
          label: 'Lista',
        ),
        const BottomNavigationBarItem(
          icon: Icon(Icons.logout),
          label: 'Deslogar',
        ),
      ],
      onTap: (int index) {
        if (index == 1) {
          FirebaseAuth.instance.signOut();
          Navigator.of(context).push<dynamic>(MaterialPageRoute<dynamic>(
            builder: (_) => const WelcomeScreen(),
          ));
        }
      },
    ),
  );
```

```
  }
}
::::::::::::::
ui/screens/tenant/common/bottom_navigation_bar/common_bottom_navigation_bar.dart
::::::::::::::
import 'package:flutter/material.dart';
import 'package:mobile/ui/screens/tenant/favorites/favorites_screen.dart';
import 'package:mobile/ui/screens/tenant/list/list_screen.dart';
import 'package:mobile/ui/screens/tenant/map/map_screen.dart';
import 'package:mobile/ui/screens/welcome/welcome_screen.dart';

enum BottomNavigationIndex {
  MAP,
  LIST,
  FAVORITES,
  LOGOUT,
}

class CommonBottomNavigation extends StatelessWidget {
  const CommonBottomNavigation({Key? key, required this.currentIndex})
      : super(key: key);

  final BottomNavigationIndex currentIndex;

  void _onTap(int index, BuildContext context) {
    late Widget nextPage;

    switch (BottomNavigationIndex.values[index]) {
      case BottomNavigationIndex.MAP:
        nextPage = const MapScreen();
        break;
      case BottomNavigationIndex.LIST:
        nextPage = const ListScreen();
        break;
      case BottomNavigationIndex.FAVORITES:
        nextPage = const FavoritesScreen();
        break;
```

```
    case BottomNavigationIndex.LOGOUT:
      nextPage = const WelcomeScreen();
      break;
  }

  // Push next page without animation
  Navigator.of(context).push<dynamic>(PageRouteBuilder<dynamic>(
    pageBuilder: (_, __, ___) => nextPage,
    transitionDuration: Duration.zero,
  ));
}

@override
Widget build(BuildContext context) {
  return BottomNavigationBar(
    currentIndex: currentIndex.index,
    items: <BottomNavigationBarItem>[
      const BottomNavigationBarItem(
        icon: Icon(Icons.map),
        label: 'Mapa',
      ),
      const BottomNavigationBarItem(
        icon: Icon(Icons.list),
        label: 'Lista',
      ),
      const BottomNavigationBarItem(
        icon: Icon(Icons.favorite),
        label: 'Favoritos',
      ),
      // const BottomNavigationBarItem(
      //   icon: Icon(Icons.logout),
      //   label: 'Sair',
      // ),
    ],
    onTap: (int index) => _onTap(index, context),
  );
}
```

```
}
::::::::::::::
ui/screens/tenant/common/dropdown/department_dropdown.dart
::::::::::::::
import 'package:flutter/material.dart';
import 'package:mobile/core/tenant/departments/department_provider.dart';
import 'package:provider/provider.dart';

class DepartmentDropdown extends StatelessWidget {
  const DepartmentDropdown({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return InputDecorator(
      decoration: const InputDecoration(
        icon: Icon(Icons.school),
        labelText: 'Centro de estudos',
      ),
      child: DropdownButtonHideUnderline(
        child: Consumer<DepartmentProvider>(
          builder: (_, DepartmentProvider provider, __) =>
              DropdownButton<String>(
            isDense: true,
            value: provider.selected,
            onChanged: (String? department) {
              provider.selected = department;
            },
            items: provider.departments
                .map((String department) => DropdownMenuItem<String>(
                      value: department,
                      child: Text(department),
                    ))
                .toList(),
          ),
        ),
      ),
    );
```

```
  }
}
::::::::::::::
ui/screens/tenant/details/details_body.dart
::::::::::::::
import 'package:flutter/material.dart';
import 'package:flutter_widget_from_html_core/flutter_widget_from_html_core.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/core/tenant/favorites/favorites_provider.dart';
import 'package:mobile/ui/common/listing/listing_item_image_slider.dart';
import 'package:mobile/ui/common/listing/listing_item_info.dart';
import 'package:mobile/ui/screens/tenant/details/details_prices.dart';
import 'package:url_launcher/url_launcher.dart';

class DetailsBody extends StatelessWidget {
  const DetailsBody(this.listing, this.favorites, {Key? key}) : super(key: key);

  final Listing listing;
  final FavoritesProvider favorites;

  List<Widget> getContacts() {
    return listing.phoneNumbers.map((dynamic phoneNumber) {
      final String phone = phoneNumber as String;

      return TextButton.icon(
        icon: const Icon(Icons.phone),
        label: Text(phone),
        onPressed: () async {
          final String url = 'tel:$phone';
          if (await canLaunch(url)) {
            await launch(url);
          } else {
            throw 'Could not launch $url';
          }
        },
      );
    }).toList();
```

```
}

@override
Widget build(BuildContext context) => SingleChildScrollView(
    child: Column(
      children: <Widget>[
        ListingItemImageSlider(listing.picturesUrls),
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: <Widget>[
              Text(
                'Descrição',
                style: Theme.of(context).textTheme.headline6,
              ),
              HtmlWidget(listing.description),
              const SizedBox(height: 8),
              Text(
                'Valores',
                style: Theme.of(context).textTheme.headline6,
              ),
              DetailsPrices(listing),
              const SizedBox(height: 8),
              Text(
                'Características',
                style: Theme.of(context).textTheme.headline6,
              ),
              ListingItemInfo(listing),
              const SizedBox(height: 8),
              Text(
                'Contatos',
                style: Theme.of(context).textTheme.headline6,
              ),
              Column(
                children: getContacts(),
              ),
```

```
              ],
            ),
          )
        ],
      ),
    );
}
::::::::::::::
ui/screens/tenant/details/details_prices.dart
::::::::::::::
import 'package:flutter/material.dart';
import 'package:mobile/core/models/listing_model.dart';

class DetailsPrices extends StatelessWidget {
  const DetailsPrices(this.listing, {Key? key}) : super(key: key);

  final Listing listing;

  @override
  Widget build(BuildContext context) => Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: <Column>[
          Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: <Widget>[
              const Text('Aluguel: '),
              const Text('Condomínio: '),
              const Text('IPTU (mensal): '),
              Text(
                'Valor total: ',
                style: Theme.of(context).textTheme.bodyText1,
              ),
            ],
          ),
          Column(
            crossAxisAlignment: CrossAxisAlignment.end,
            children: <Widget>[
```

```
            Text('R\$ ${listing.priceMonthly}'),
            Text('R\$ ${listing.condoFeeMonthly}'),
            Text('R\$ ${listing.propertyTaxMonthly}'),
            Text(
              'R\$ ${listing.totalPrice.toString()}',
              style: Theme.of(context).textTheme.bodyText1,
            ),
          ],
        ),
      ],
    );
}
::::::::::::::
ui/screens/tenant/details/details_screen.dart
::::::::::::::
import 'dart:developer';

import 'package:flutter/material.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/core/tenant/favorites/favorites_provider.dart';
import
'package:mobile/ui/screens/tenant/common/bottom_navigation_bar/common_bottom_naviga
tion_bar.dart';
import 'package:mobile/ui/screens/tenant/details/details_body.dart';
import 'package:provider/provider.dart';

class DetailsScreen extends StatelessWidget {
  const DetailsScreen(this.listing, {Key? key}) : super(key: key);

  final Listing listing;

  @override
  Widget build(BuildContext context) {
    return Consumer<FavoritesProvider>(
      builder: (_, FavoritesProvider favorites, __) => Scaffold(
        appBar: AppBar(
          title: Text(listing.title),
```

```
        ),
        body: DetailsBody(listing, favorites),
        bottomNavigationBar: const CommonBottomNavigation(
          currentIndex: BottomNavigationIndex.LIST,
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: () {
            if (favorites.contains(listing)) {
              log('Listing ${listing.id} removido dos favoritos!');
              favorites.remove(listing);
            } else {
              log('Listing ${listing.id} adicionado aos favoritos!');
              favorites.add(listing);
            }
          },
          child: Icon(favorites.contains(listing)
              ? Icons.favorite
              : Icons.favorite_border),
        ),
      ),
    );
  }
}
```

::::::::::::::

ui/screens/tenant/favorites/favorites_body.dart

::::::::::::::

```
import 'package:flutter/material.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/ui/common/listing/listing_item.dart';

class FavoritesBody extends StatefulWidget {
  const FavoritesBody({Key? key, required this.listings}) : super(key: key);

  final List<Listing> listings;

  @override
  _FavoritesBodyState createState() => _FavoritesBodyState();
```

```
}

class _FavoritesBodyState extends State<FavoritesBody> {
  @override
  Widget build(BuildContext context) {
    if (widget.listings.isEmpty) {
      return const Center(child: Text('Nenhum anúncio favoritado'));
    }

    return ListView.builder(
      itemCount: widget.listings.length,
      itemBuilder: (BuildContext context, int index) =>
          ListingItem(widget.listings[index]),
    );
  }
}
```

::::::::::::::

ui/screens/tenant/favorites/favorites_screen.dart

::::::::::::::

```
import 'package:flutter/material.dart';
import 'package:mobile/core/tenant/departments/department_provider.dart';
import 'package:mobile/core/tenant/favorites/filtered_favorites_provider.dart';
import 'package:mobile/core/tenant/filters/filter_provider.dart';
import
'package:mobile/ui/screens/tenant/common/bottom_navigation_bar/common_bottom_naviga
tion_bar.dart';
import 'package:mobile/ui/screens/tenant/favorites/favorites_body.dart';
import 'package:mobile/ui/screens/tenant/filters/filters_screen.dart';
import 'package:mobile/ui/screens/tenant/map/filters.dart';
import 'package:provider/provider.dart';

class FavoritesScreen extends StatelessWidget {
  const FavoritesScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```
appBar: AppBar(
  automaticallyImplyLeading: false,
  title: Consumer<DepartmentProvider>(
    builder: (_, DepartmentProvider department, __) =>
        Text(department.selected!),
  ),
  actions: <IconButton>[
    IconButton(
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute<FiltersScreen>(
                builder: (_) => const FiltersScreen()),
          );
        },
        icon: const Icon(Icons.filter_alt))
  ],
  bottom: PreferredSize(
    child: Consumer<FilterProvider>(
        builder: (_, FilterProvider provider, __) =>
            Filters(provider: provider)),
    preferredSize: const Size.fromHeight(40.0),
  ),
),
body: Consumer<FilteredFavoritesProvider>(
  builder: (_, FilteredFavoritesProvider favorites, __) {
    return FavoritesBody(
      listings: favorites.listings,
      key: Key(
        favorites.listings.length.toString(), // FIXME
      ),
    );
  },
),
bottomNavigationBar: const CommonBottomNavigation(
  currentIndex: BottomNavigationIndex.FAVORITES,
),
```

```
    );
  }
}
::::::::::::::
ui/screens/tenant/filters/filters_screen.dart
::::::::::::::
import 'dart:math';

import 'package:flutter/material.dart';
import 'package:mobile/core/models/address_model.dart';
import 'package:mobile/core/tenant/filters/filter_provider.dart';
import 'package:mobile/ui/screens/tenant/common/dropdown/department_dropdown.dart';
import 'package:provider/provider.dart';

class FiltersScreen extends StatelessWidget {
  const FiltersScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Consumer<FilterProvider>(
      builder: (_, FilterProvider filters, __) => Scaffold(
        appBar: AppBar(
          title: const Text('Filtros'),
          actions: <Widget>[
            IconButton(
              onPressed: filters.clear,
              icon: const Icon(Icons.clear),
            )
          ],
        ),
        body: SingleChildScrollView(
          child: Padding(
            padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 20),
            child: Column(
              children: <Widget>[
                const DepartmentDropdown(),
                InputDecorator(
```

```
  decoration: const InputDecoration(
    icon: Icon(Icons.sort),
    labelText: 'Ordenar por',
  ),
  child: DropdownButtonHideUnderline(
    child: DropdownButton<SortBy>(
      isDense: true,
      value: filters.sortBy,
      onChanged: (SortBy? sortBy) {
        filters.sortBy = sortBy ?? SortBy.PRICE;
      },
      items: SortBy.values.map((SortBy value) {
        late String text;

        switch (value) {
          case SortBy.PRICE:
            text = 'Menor preço';
            break;
          case SortBy.WALKING:
            text = 'Menor distância caminhada';
            break;
          case SortBy.BICYCLE:
            text = 'Menor distância de bicicleta';
            break;
        }

        return DropdownMenuItem<SortBy>(
          value: value,
          child: Text(text),
        );
      }).toList(),
    ),
  ),
),
TextFormField(
  initialValue: filters.priceRange.y.toString(),
  keyboardType: TextInputType.number,
```

```
  decoration: const InputDecoration(
    icon: Icon(Icons.attach_money),
    labelText: 'Preço máximo',
    suffixText: r'R$',
  ),
  onFieldSubmitted: (String? value) {
    filters.priceRange =
        Point<int>(filters.priceRange.x, int.parse(value!));
  },
  validator: (String? value) {
    return value == null || int.tryParse(value) == null
        ? 'Apenas números'
        : null;
  },
),
TextFormField(
  initialValue: filters.maxDistanceBicycle.inMinutes.toString(),
  keyboardType: TextInputType.number,
  decoration: const InputDecoration(
    icon: Icon(Icons.directions_bike),
    labelText: 'Distância máxima de bicicleta',
    suffixText: 'min',
  ),
  onFieldSubmitted: (String? value) {
    filters.maxDistanceBicycle =
        Duration(minutes: int.parse(value!));
  },
  validator: (String? value) {
    return value == null || int.tryParse(value) == null
        ? 'Apenas números'
        : null;
  },
),
TextFormField(
  initialValue: filters.maxDistanceWalking.inMinutes.toString(),
  keyboardType: TextInputType.number,
  decoration: const InputDecoration(
```

```
      icon: Icon(Icons.directions_walk),
      labelText: 'Distância máxima a pé',
      suffixText: 'min',
    ),
    onFieldSubmitted: (String? value) {
      filters.maxDistanceWalking =
          Duration(minutes: int.parse(value!));
    },
    validator: (String? value) {
      return value == null || int.tryParse(value) == null
          ? 'Apenas números'
          : null;
    },
  ),
  TextFormField(
    initialValue: filters.minSquareMeters.toString(),
    keyboardType: TextInputType.number,
    decoration: const InputDecoration(
      icon: Icon(Icons.photo_size_select_small),
      labelText: 'Área mínima',
      suffixText: 'm2',
    ),
    onFieldSubmitted: (String? value) {
      filters.minSquareMeters =
          value != null ? int.parse(value) : 0;
    },
    validator: (String? value) {
      return value == null || int.tryParse(value) == null
          ? 'Apenas números'
          : null;
    },
  ),
  TextFormField(
    initialValue: filters.minBedrooms.toString(),
    keyboardType: TextInputType.number,
    decoration: const InputDecoration(
      icon: Icon(Icons.bed),
```

```
    labelText: 'Número mínimo de quartos',
  ),
  onFieldSubmitted: (String? value) {
    filters.minBedrooms = value != null ? int.parse(value) : 0;
  },
  validator: (String? value) {
    return value == null || int.tryParse(value) == null
        ? 'Apenas números'
        : null;
  },
),
const SizedBox(height: 16),
Row(
  children: <Widget>[
    const Text('Tipo de imóvel:'),
  ],
),
FractionallySizedBox(
  widthFactor: 1,
  child: Wrap(spacing: 8, children: <Widget>[
    ChoiceChip(
      label: const Text('Kitnet'),
      avatar: const Icon(Icons.other_houses),
      selected: filters.isKitnet,
      onSelected: (bool selected) {
        filters.isKitnet = selected;
      },
    ),
    ChoiceChip(
      label: const Text('Apartamento'),
      avatar: const Icon(Icons.apartment),
      selected: filters.isApartment,
      onSelected: (bool selected) {
        filters.isApartment = selected;
      },
    ),
    ChoiceChip(
```

```
      label: const Text('Casa'),
      avatar: const Icon(Icons.house),
      selected: filters.isHouse,
      onSelected: (bool selected) {
        filters.isHouse = selected;
      },
    ),
  ]),
),
const SizedBox(height: 16),
Row(
  children: <Widget>[
    const Text('Bairros:'),
  ],
),
FractionallySizedBox(
  widthFactor: 1,
  child: Wrap(
    spacing: 8,
    children: <Widget>[
      ChoiceChip(
        label: const Text('Pantanal'),
        avatar: const Icon(Icons.location_on),
        selected: filters
            .isNeighborhoodSelected(Neighborhood.PANTANAL),
        onSelected: (bool selected) {
          filters.setNeighborhoods(
              Neighborhood.PANTANAL, selected);
        },
      ),
      ChoiceChip(
        label: const Text('Trindade'),
        avatar: const Icon(Icons.location_on),
        selected: filters
            .isNeighborhoodSelected(Neighborhood.TRINDADE),
        onSelected: (bool selected) {
          filters.setNeighborhoods(
```

```
            Neighborhood.TRINDADE, selected);
        },
      ),
      ChoiceChip(
        label: const Text('Carvoeira'),
        avatar: const Icon(Icons.location_on),
        selected: filters
            .isNeighborhoodSelected(Neighborhood.CARVOEIRA),
        onSelected: (bool selected) {
          filters.setNeighborhoods(
              Neighborhood.CARVOEIRA, selected);
        },
      ),
      ChoiceChip(
        label: const Text('Córrego Grande'),
        avatar: const Icon(Icons.location_on),
        selected: filters.isNeighborhoodSelected(
            Neighborhood.CORREGO_GRANDE),
        onSelected: (bool selected) {
          filters.setNeighborhoods(
              Neighborhood.CORREGO_GRANDE, selected);
        },
      ),
      ChoiceChip(
        label: const Text('Saco dos Limões'),
        avatar: const Icon(Icons.location_on),
        selected: filters.isNeighborhoodSelected(
            Neighborhood.SACO_DOS_LIMOES),
        onSelected: (bool selected) {
          filters.setNeighborhoods(
              Neighborhood.SACO_DOS_LIMOES, selected);
        },
      ),
    ],
  ),
),
const SizedBox(height: 16),
```

```
Row(
  children: <Widget>[
    const Text('Comodidades:'),
  ],
),
FractionallySizedBox(
  widthFactor: 1,
  child: Wrap(
    spacing: 8,
    children: <Widget>[
      ChoiceChip(
        label: const Text('Ar condicionado'),
        avatar: const Icon(Icons.ac_unit),
        selected: filters.hasAirConditioning,
        onSelected: (bool selected) {
          filters.hasAirConditioning = selected;
        },
      ),
      ChoiceChip(
        label: const Text('Estacionamento'),
        avatar: const Icon(Icons.local_parking),
        selected: filters.hasParking,
        onSelected: (bool selected) {
          filters.hasParking = selected;
        },
      ),
      ChoiceChip(
        label: const Text('Mobiliado'),
        avatar: const Icon(Icons.bed),
        selected: filters.isFurnished,
        onSelected: (bool selected) {
          filters.isFurnished = selected;
        },
      ),
      ChoiceChip(
        label: const Text('Lavanderia'),
        selected: filters.hasLaundry,
```

```
  avatar: const Icon(Icons.local_laundry_service),
  onSelected: (bool selected) {
    filters.hasLaundry = selected;
  },
),
ChoiceChip(
  label: const Text('Piscina'),
  avatar: const Icon(Icons.pool),
  selected: filters.hasPool,
  onSelected: (bool selected) {
    filters.hasPool = selected;
  },
),
ChoiceChip(
  label: const Text('Churrasqueira'),
  avatar: const Icon(Icons.outdoor_grill),
  selected: filters.hasGrill,
  onSelected: (bool selected) {
    filters.hasGrill = selected;
  },
),
ChoiceChip(
  label: const Text('Elevador'),
  avatar: const Icon(Icons.elevator),
  selected: filters.hasElevator,
  onSelected: (bool selected) {
    filters.hasElevator = selected;
  },
),
ChoiceChip(
  label: const Text('Academia'),
  avatar: const Icon(Icons.fitness_center),
  selected: filters.hasGym,
  onSelected: (bool selected) {
    filters.hasGym = selected;
  },
),
```

```
                    ChoiceChip(
                      label: const Text('Pets'),
                      avatar: const Icon(Icons.pets),
                      selected: filters.petsAllowed,
                      onSelected: (bool selected) {
                        filters.petsAllowed = selected;
                      },
                    ),
                  ],
                ),
              ),
            ],
          ),
        ),
      ),
    ),
  );
 }
}
::::::::::::::
ui/screens/tenant/list/list_body.dart
::::::::::::::
import 'package:flutter/material.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/ui/common/listing/listing_item.dart';

class ListScreenBody extends StatefulWidget {
  const ListScreenBody({Key? key, required this.listings}) : super(key: key);

  final List<Listing> listings;

  @override
  _ListScreenBodyState createState() => _ListScreenBodyState();
}

class _ListScreenBodyState extends State<ListScreenBody> {
  @override
```

```
  Widget build(BuildContext context) {
    if (widget.listings.isEmpty) {
      return const Center(child: Text('Nenhum anúncio encontrado'));
    }

    return ListView.builder(
      itemCount: widget.listings.length,
      itemBuilder: (BuildContext context, int index) =>
          ListingItem(widget.listings[index]),
    );
  }
}
```

::::::::::::::

ui/screens/tenant/list/list_screen.dart

::::::::::::::

```
import 'package:flutter/material.dart';
import 'package:mobile/core/tenant/departments/department_provider.dart';
import 'package:mobile/core/tenant/filters/filter_provider.dart';
import 'package:mobile/core/tenant/listings/filtered_listings_provider.dart';
import
'package:mobile/ui/screens/tenant/common/bottom_navigation_bar/common_bottom_naviga
tion_bar.dart';
import 'package:mobile/ui/screens/tenant/filters/filters_screen.dart';
import 'package:mobile/ui/screens/tenant/list/list_body.dart';
import 'package:mobile/ui/screens/tenant/map/filters.dart';
import 'package:provider/provider.dart';

class ListScreen extends StatelessWidget {
  const ListScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        title: Consumer<DepartmentProvider>(
          builder: (_, DepartmentProvider department, __) =>
```

```
            Text(department.selected!),
        ),
        actions: <IconButton>[
          IconButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute<FiltersScreen>(
                      builder: (_) => const FiltersScreen()),
                );
              },
              icon: const Icon(Icons.filter_alt))
        ],
        bottom: PreferredSize(
          child: Consumer<FilterProvider>(
              builder: (_, FilterProvider provider, __) =>
                  Filters(provider: provider)),
          preferredSize: const Size.fromHeight(40.0),
        ),
      ),
      body: Consumer<FilteredListingsProvider>(
        builder: (_, FilteredListingsProvider provider, __) {
          return ListScreenBody(
            listings: provider.listings,
            key: Key(
              provider.listings.length.toString(), // FIXME
            ),
          );
        },
      ),
      bottomNavigationBar: const CommonBottomNavigation(
        currentIndex: BottomNavigationIndex.LIST,
      ),
    );
  }
}
::::::::::::::
```

ui/screens/tenant/map/filters.dart

```
::::::::::::::
import 'package:flutter/material.dart';

import 'package:mobile/core/tenant/filters/filter_provider.dart';
import 'package:mobile/ui/screens/tenant/filters/filters_screen.dart';

class Filters extends StatelessWidget {
  const Filters({Key? key, required this.provider}) : super(key: key);

  final FilterProvider provider;

  @override
  Widget build(BuildContext context) => InkWell(
        onTap: () {
          Navigator.push(
            context,
            MaterialPageRoute<FiltersScreen>(
                builder: (_) => const FiltersScreen()),
          );
        },
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: <Widget>[
            Chip(
              avatar: const Icon(Icons.attach_money),
              label: Text('< ${provider.priceRange.y}'),
            ),
            Chip(
              avatar: const Icon(Icons.directions_bike),
              label: Text('< ${provider.maxDistanceBicycle.inMinutes} min'),
            ),
            Chip(
              avatar: const Icon(Icons.directions_walk),
              label: Text('< ${provider.maxDistanceWalking.inMinutes} min'),
            )
          ],
```

```
        ),
      );
}
::::::::::::::
ui/screens/tenant/map/map_body.dart
::::::::::::::
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:mobile/core/models/listing_model.dart';
import 'package:mobile/ui/screens/tenant/details/details_screen.dart';

class MapBody extends StatefulWidget {
  const MapBody({Key? key, required this.listings, required this.department})
      : super(key: key);

  final List<Listing> listings;
  final LatLng department;

  @override
  _MapBodyState createState() => _MapBodyState();
}

class _MapBodyState extends State<MapBody> {
  final Completer<GoogleMapController> _controller =
      Completer<GoogleMapController>();

  final CameraPosition _kUFSC = const CameraPosition(
    target: LatLng(-27.600338308488425, -48.51757616890526),
    zoom: 14,
  );

  late Set<Marker> markers;

  void navigateToListing(Listing listing) {
    Navigator.of(context).push<dynamic>(MaterialPageRoute<dynamic>(
```

```
      builder: (_) => DetailsScreen(
        widget.listings.firstWhere((Listing item) => item.id == listing.id),
      ),
    ));
  }

  @override
  void initState() {
    super.initState();

    markers = widget.listings
        .map((Listing listing) => Marker(
            markerId: MarkerId(listing.id),
            position: listing.address.position,
            onTap: () => navigateToListing(listing)))
        .toSet();

    markers.add(Marker(
      markerId: const MarkerId('department'),
      position: widget.department,
      icon: BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueYellow),
    ));
  }

  @override
  Widget build(BuildContext context) => GoogleMap(
        markers: markers,
        mapType: MapType.normal,
        initialCameraPosition: _kUFSC,
        onMapCreated: (GoogleMapController controller) {
          _controller.complete(controller);
        },
      );
}
```

::::::::::::::

ui/screens/tenant/map/map_screen.dart

::::::::::::::

```
import 'package:flutter/material.dart';
import 'package:mobile/core/tenant/departments/department_provider.dart';
import 'package:mobile/core/tenant/filters/filter_provider.dart';
import 'package:mobile/core/tenant/listings/filtered_listings_provider.dart';
import
'package:mobile/ui/screens/tenant/common/bottom_navigation_bar/common_bottom_naviga
tion_bar.dart';
import 'package:mobile/ui/screens/tenant/filters/filters_screen.dart';
import 'package:mobile/ui/screens/tenant/map/map_body.dart';
import 'package:provider/provider.dart';

import 'filters.dart';

class MapScreen extends StatelessWidget {
  const MapScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        title: Consumer<DepartmentProvider>(
          builder: (_, DepartmentProvider department, __) =>
              Text(department.selected!),
        ),
        actions: <IconButton>[
          IconButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute<FiltersScreen>(
                      builder: (_) => const FiltersScreen()),
                );
              },
              icon: const Icon(Icons.filter_alt))
        ],
        bottom: PreferredSize(
```

```
          child: Consumer<FilterProvider>(
              builder: (_, FilterProvider provider, __) =>
                  Filters(provider: provider)),
          preferredSize: const Size.fromHeight(40.0),
        ),
      ),
      body: Consumer2<FilteredListingsProvider, DepartmentProvider>(
        builder: (_, FilteredListingsProvider listingsProvider,
            DepartmentProvider departmentProvider, __) {
          return MapBody(
            listings: listingsProvider.listings,
            department: departmentProvider.position,
            key: Key(
              listingsProvider.listings.length.toString(), // FIXME
            ),
          );
        },
      ),
      bottomNavigationBar: const CommonBottomNavigation(
        currentIndex: BottomNavigationIndex.MAP,
      ),
    );
  }
}
```

::::::::::::::

ui/screens/welcome/welcome_screen.dart

::::::::::::::

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:mobile/ui/screens/landlord/auth_gate/auth_gate_screen.dart';
import 'package:mobile/ui/screens/landlord/list/list_screen.dart';
import 'package:mobile/ui/screens/tenant/common/dropdown/department_dropdown.dart';

class WelcomeScreen extends StatefulWidget {
  const WelcomeScreen({Key? key}) : super(key: key);

  @override
```

```
  State<WelcomeScreen> createState() => _WelcomeScreenState();
}

class _WelcomeScreenState extends State<WelcomeScreen> {
  User? user;

  @override
  void initState() {
    super.initState();

    FirebaseAuth.instance.authStateChanges().listen((User? event) {
      user = event;
    });
  }

  @override
  Widget build(BuildContext context) => Scaffold(
        body: Padding(
          padding: const EdgeInsets.all(32.0),
          child: Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.spaceAround,
              crossAxisAlignment: CrossAxisAlignment.center,
              children: <Widget>[
                Text(
                  'Bem-vindo',
                  style: Theme.of(context).textTheme.headline3,
                ),
                Column(
                  children: <Widget>[
                    const Text('Selecione seu centro de estudos:'),
                    const SizedBox(height: 8),
                    const DepartmentDropdown(),
                  ],
                ),
                TextButton(
                  child: const Text('Quero anunciar meu imóvel'),
```

```
                onPressed: () {
                  Widget nextScreen;

                  if (user != null) {
                    nextScreen = const LandlordListScreen();
                  } else {
                    nextScreen = const AuthGateScreen();
                  }

                  Navigator.of(context)
                      .push<dynamic>(MaterialPageRoute<dynamic>(
                    builder: (_) => nextScreen,
                  ));
                },
              )
            ],
          ),
        ),
      ),
    );
}
```