INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - CÂMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

EMANUEL GOBI LUZIA

LAUNDRYAPP: APLICATIVO DE LAVAGEM DE ROUPA

PASSO FUNDO
2020

EMANUEL GOBI LUZIA

LAUNDRYAPP: APLICATIVO DE LAVAGEM DE ROUPA

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, Campus Passo Fundo, como requisito parcial para a aprovação na disciplina de Projeto de Conclusão II (PC II).

Orientador: Jorge Luis Bavaresco

PASSO FUNDO

2020

Agradecimentos

Gostaria de agradecer primeiramente a Deus, por ter me dado a oportunidade de estar nesse curso, que permitiu que eu conhecesse diversas pessoas, com qual fiz muitas amizades, que influenciaram até mesmo em minha carreira profissional.

Agradeço também a minha família que me apoiou e me deu forças em todos os momentos da minha vida, um agradecimento em especial ao professor Jorge Luis Boeira Bavaresco, que sempre demonstrou bom ânimo e me deu apoio durante o desenvolvimento deste trabalho.

Por fim agradeço a todos professores que tive contato durante o curso, através de seus ensinamentos, me tornei um estudante melhor, um profissional melhor, e uma pessoa melhor.

Resumo

Atualmente muitos aplicativos de troca de serviços vêm se tornando cada vez mais comuns na vida das pessoas. Desde aplicativos relacionados a transporte, alimentação, locação de imóveis, entre outros. Baseado nisso constatou-se que no âmbito nacional, não existe um aplicativo relacionado ao serviço de lavagem de roupas, que conecta o cliente diretamente com o fornecedor de serviço, seguindo assim os preceitos estabelecidos pela economia compartilhada. Baseado nisso foi construído um aplicativo que permite a conexão entre clientes e fornecedores de serviços relacionados a lavagem de roupa. Para o desenvolvimento do trabalho, foi utilizado o Flutter em conjunto com o Firebase, ambos são softwares pertencentes a Google. O resultado obtido foi uma aplicação onde o acesso a telas e ações são controladas de acordo com o tipo do usuário logado (cliente ou fornecedor de serviço), caso o usuário seja um cliente ele tem a possibilidade de criar pedidos com as peças de roupas e também de selecionar os fornecedores de serviços mais próximos a ele ou que apresentem um valor mais barato pelo serviço de lavagem de roupas.

Palavras-chave: Aplicativo. Flutter. Lavagem. Roupa. Firebase

Abstract

Today many service exchange applications are becoming more and more common in people's lives. From applications related to transportation, food, rental properties, among others. Based on this it was found that at the national level, there is no application related to the laundry service, which connects the customer directly with the service provider, thus following the precepts established by the shared economy. Based on this, an application was built that allows the connection between customers and service providers related to laundry. For the development of the work, Flutter was used in conjunction with Firebase, both are software belonging to Google. The result obtained was an application where access to screens and actions are controlled according to the type of user logged in (customer or service provider), if the user is a customer he has the possibility to create orders with the pieces of clothing and also to select the service providers closest to him or who have a cheaper value for the laundry service.

Keywords: Application. Flutter. Wash. Clothes. Firebase

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES E DE SIGLAS

ANEL - Associação Nacional das Empresas de Lavanderia

SDK - Software Development Kit (Kit de Desenvolvimento de Software)

UI - User Interface (Design de interface de usuário)

SUMÁRIO.

1 INTRODUÇÃO

Com a democratização da Internet e dos Smartphones, conceitos como a economia compartilhada tornam-se cada vez mais difundidos na sociedade. Um exemplo prático disso são as empresas, tais quais Uber, eBay e Airbnb, que usam amplamente esse conceito. Nesse sentido, a economia compartilhada se dedica a conectar o cliente diretamente com o prestador de serviço (SIGNES,2017).

Constatou-se que entre os diversos serviços existentes hoje que utilizam a economia compartilhada no âmbito nacional, não há um que seja voltado para o serviço de lavagem de roupa e respeite os preceitos da economia compartilhada. Entretanto, segundo a ANEL (2018), em 10 anos o setor de lavanderias deve crescer mais de 100%, fato que demonstra o quanto esse mercado pode ser explorado utilizando esse conceito.

Assim foi desenvolvido um aplicativo mobile multiplataforma de prestação de serviços de lavagem de roupas, utilizando a tecnologia Flutter como ferramenta de desenvolvimento, o aplicativo visa ligar o tomador do serviço diretamente com o prestador do serviço, utilizando assim o conceito da economia compartilhada.

1.1 OBJETIVOS

Nesta seção serão apresentados os objetivos gerais e específicos do projeto.

1.1.1 Objetivo geral

Desenvolver um aplicativo multiplataforma de prestação de serviços relacionados a lavagem de roupa.

1.1.2 Objetivos específicos

- Possibilitar cadastros dos usuários ao sistema;
- Realizar a conexão dos prestadores e os tomadores de serviço;
- Realizar Diagramação do caso de uso e das classes;

- Utilizar as ferramentas *Flutter* e *FireBase* para realizar o trabalho;
- Explorar e demonstrar a utilização da gerência de estado no *Flutter;*

2 REFERENCIAL TEÓRICO

Nos próximos tópicos serão apresentadas as tecnologias e conceitos que foram utilizados na realização desse trabalho.

2.1 Flutter

Flutter é um SDK desenvolvida pelo Google que surgiu com a ideia de proporcionar aos desenvolvedores a criação de aplicações mobile multiplataforma, que compilariam o código de forma nativa. A razão desta tecnologia ter sido escolhida para a realização do presente trabalho é pelo fato do Flutter ser uma tecnologia nova e promissora no mercado. Atualmente grandes empresas como NUBANK, Alibaba e ebay (Flutter,2020), já desenvolveram seus aplicativos utilizando essa ferramenta. O Flutter usa a linguagem de programação Dart, essa linguagem permite que a aplicação seja compilada diretamente para a linguagem de máquina, parece, pois, que isso caracteriza-se como uma vantagem sobre outros frameworks como React Native, o qual necessita de uma “ponte” para compilar o código, isso acaba afetando diretamente o desempenho da aplicação. Outra característica que a linguagem Dart proporciona é o “*Hot Reload*”, que possui a capacidade de atualizar e aplicar em menos de um segundo alterações que foram feitas na aplicação, ou seja, qualquer modificação no código pode ser visto na tela com milissegundos de diferenças, sem precisar reiniciar toda a aplicação para que as mudanças sejam aplicadas.

Em outros Frameworks de desenvolvimento mobile multiplataforma, como IONIC React Native, entre outros são utilizados os componentes nativos para cada plataforma, isto é, um botão na plataforma Android será diferente de um botão na plataforma iOS; porém no Flutter isso não ocorre pois todos os seus UI são desenhados pelo seu mecanismo gráfico, não há camada de comunicação entre a View e seu código, por conta disso a aplicação alcança um melhor desempenho.

2.2 FireBase

O FireBase é um serviço de back-end para aplicações web e mobile que oferece várias funcionalidades, tais quais banco de dados, ferramentas de análises, relatórios de erros, escalabilidade automática da aplicação, etc. O conceito do Firebase é que os programadores possam passar mais tempo pensando no desenvolvimento e funcionalidades da aplicação enquanto ele cuida do *back-end*, de modo que o desenvolvedor não precise se preocupar com a infraestrutura necessária para sua aplicação, eliminando também o conhecimento necessário para gerenciar isso, uma vez que o Firebase disponibiliza ferramentas que auxiliam no gerenciamento da aplicação.

O FireBase foi adquirido pelo Google em 2014 e pertence ao Google Cloud Platform (O serviço de computação em nuvem do Google). Dentro do FireBase alguns serviços disponibilizados são gratuitos, outros são grátis até determinado nível de uso; porém existem alguns que são exclusivamente pagos (Freitas,2019). De maneira geral os produtos do Firebase se dividem em dois grupos, um deles é referente ao desenvolvimento e teste do aplicativo e o outro é referente a expansão e engajamento dos usuários da aplicação. Como exemplo desses produtos podemos citar os que são referentes ao desenvolvimento e testes do aplicativo que seriam (FIREBASE, 2019): Authentication, Test Lab para Android, Crashlytics, Cloud Firestore, Test Lab para Android, Cloud Functions, Hosting, Monitoramento de desempenho e Cloud Storage. Já os produtos que seriam referentes ao engajamento do publico seriam: Cloud Messaging, Previsões, Invites, Configuração remota, Dynamic Links, AdMob e Google Analytics.

Nesse projeto foi utilizado o Authentication para realizar a autenticação do usuário no aplicativo e o Cloud Firestore, que é base dados NoSQL.

2.3 Economia Compartilhada

O conceito da economia compartilhada tornou-se popular desde a década de 1990 devido a popularização da internet e os avanços tecnológicos que viabilizaram a criação de novos modelos de negócio a partir de troca de bens e serviços entre pessoas desconhecidas. Algumas empresas acabaram consolidando esse modelo de negócio, tais quais Uber, Ifood, eBey, Airbn, etc.

A economia compartilhada pode ser dividida em 2 grupos, *crowdwork* e *on-demand*(KALIL,2017). O *Crowdwork* seria a troca de serviço ou produto no qual a distância não é um fator impeditivo para a finalização da troca ou serviço, como exemplo o eBay e Mercado Livre. Nessas duas plataformas a operação pode ser feita independente da distância entre o prestador e o tomador de serviço. Já o *on-demand, conhecido também como* crowdwork *offline* ou ainda Uberização, depende diretamente da localização, uma vez que o serviço prestado está em um contexto local, geralmente é utilizado para serviços de transporte e limpeza, podemos citar como exemplo a plataforma Uber – responsável por popularizar amplamente esse conceito.

3 METODOLOGIA

Nos próximos tópicos será apresentada a metodologia utilizada para o desenvolvimento do presente trabalho.

3.1 Análise

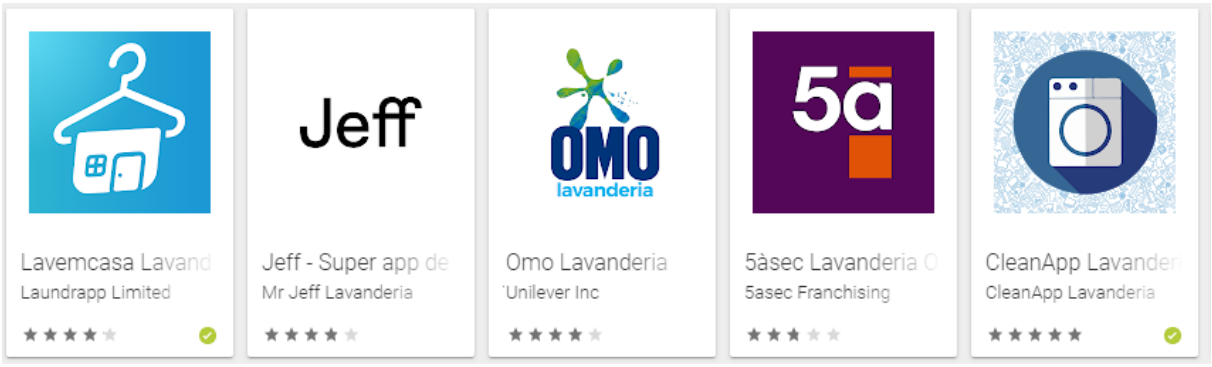
Nessa seção será realizada descrição superficial das funcionalidades desenvolvidas no aplicativo.

A aplicação possui um sistema de autenticação por usuário e senha, sendo necessária a realização de um cadastro via e-mail. Nas informações do cadastro, será necessário informar o tipo de usuário (fornecedor de serviço ou cliente) caso o tipo seja fornecedor é necessário informar a taxa que será aplicada em cima dos produtos. Caso o tipo do usuário seja cliente, em seguida será possível utilizar o sistema. Haverá uma tela que possibilitará ao cliente a busca por prestadores de serviços, os quais serão listados ao cliente, as principais informações destacadas na listagem do fornecedor são a taxa cadastrada no momento da criação da conta, e também a distância em que ele está do cliente. Após selecionar o fornecedor, o cliente poderá selecionar as roupas que deseja lavar, as quais possuem uma tabela de preço fixa dentro do sistema, será acrescentado a esse valor a taxa do fornecedor selecionado. Desse modo, após as peças de roupas serem escolhidas o cliente poderá leva-las até o fornecedor do serviço. Após a realização do serviço, o cliente deve buscar as roupas e ao indicar a finalização, o pagamento será efetivado pelo próprio aplicativo.

3.2 Aplicativos Correlacionados

Para fins de comparação, foi realizada uma busca por aplicativos do mesmo ramo com uma proposta semelhante. Para localizar os aplicativos foi utilizado como parâmetro na pesquisa do Google Play a palavra “Lavanderia”, conforme é exibido na Figura 1. Nesta pesquisa foram selecionados os aplicativos que possuíam as maiores notas de avaliação. Os aplicativos analisados foram: “Lavemcasa” (GOOGLE Play, 2019), “Omo lavanderia” (GOOGLE Play, 2019), “CleanApp Lavanderia” (GOOGLE Play, 2019) e “MrJeff” (GOOGLE Play, 2019).

Figura 1 - Capturas de tela Google Play



***Fonte**: Google Play, 2020*

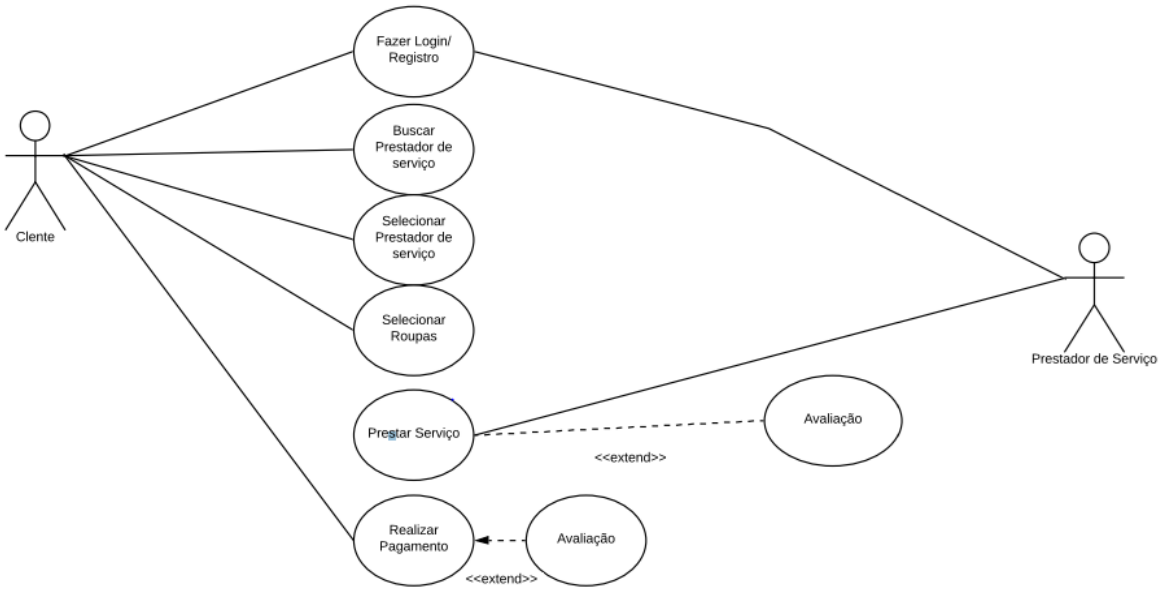
Todos esses aplicativos são franquias de lavanderias que funcionam apenas em algumas cidades com foco no serviço de lavagem de roupa por assinatura mensal ou semanal, alguns deles também possuem o serviço de buscar e devolver as roupas na casa do cliente. Nenhum dos aplicativos analisados viabiliza o contato de partes desconhecidas para realizar uma troca de serviço ou produto, assim nenhum deles utiliza o conceito da economia compartilhada apresentada neste trabalho.

3.3 Diagrama de Casos de Uso

O diagrama de casos de uso descreve uma sequência de eventos que tem por objetivo descrever as atividades realizadas por cada ator, e também facilitar o entendimento dos processos realizados pelo sistema.

O diagrama da Figura 2 descreve os casos de usos percorridos entre o Cliente e o Prestador de Serviço através do sistema.

Figura 2 - Diagrama de Casos de Uso



***Fonte**: Do autor, 2020*

O cliente e o fornecedor do serviço realizam o login ou o registro na aplicação, logo o cliente já consegue realizar a busca e a escolha do prestador de serviço, como também das roupas que deseja lavar. Por sua vez, o prestador de serviços poderá optar por realizar o trabalho.

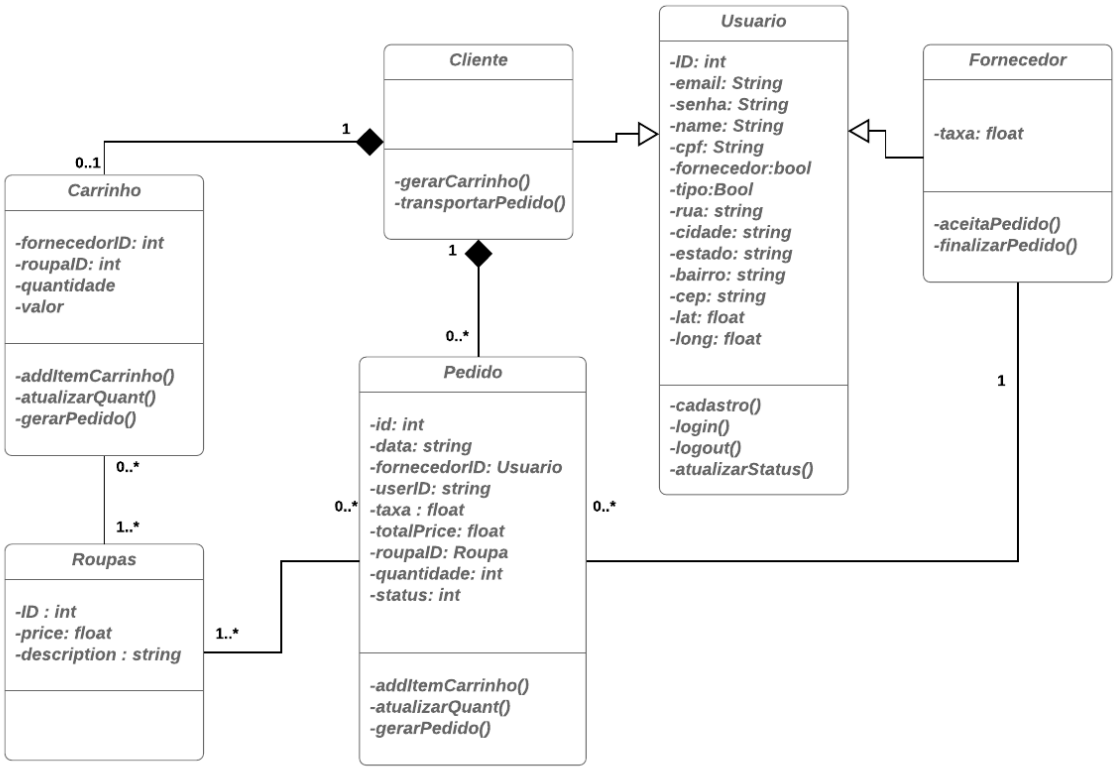
3.4 Diagrama de Classes

Na Figura 3 podemos visualizar o diagrama de classes que ilustra graficamente como será a estrutura da aplicação. Foram definidas as seguintes classes:

- **Usuário:** Armazena os dados de cadastros e os médotos comuns entre oas subclasses de Cliente e Fornecedor. Dentro dessa classe é feita a distinção do tipo de usuário, através do campo boleano ‘fornecedor’, caso o campo seja true, o usuário é um fornecedor, caso contrário é um cliente.
- **Cliente:** É uma subclasse da classe Usuário, ela armazena os metodos exclusivos do cliente, que se referem ao momento de criação do pedido.

- **Fornecedor:** Também é uma subclasse da classe Usuário, possui o campo de taxa, e alguns metodos exclusivos relacionados a troca de estado dos pedidos.
- **Carrinho:** Possui um relacionamento de composição com a Classe de cliente, o carrinho é responsável por armazenar as informações que se tornaram o pedido.
- **Pedido:** Armazena as informações como valores, produtos, data, etc. E possui também a ligação entre o cliente e o prestador de serviço selecionado.
- **Roupas:** Possui um relacionameno de composição com a Classe Pedido, é responsável por armazenar os detalhes de cada peça de roupa, como valor e quantidade.

Figura 3 - Diagrama de Classes

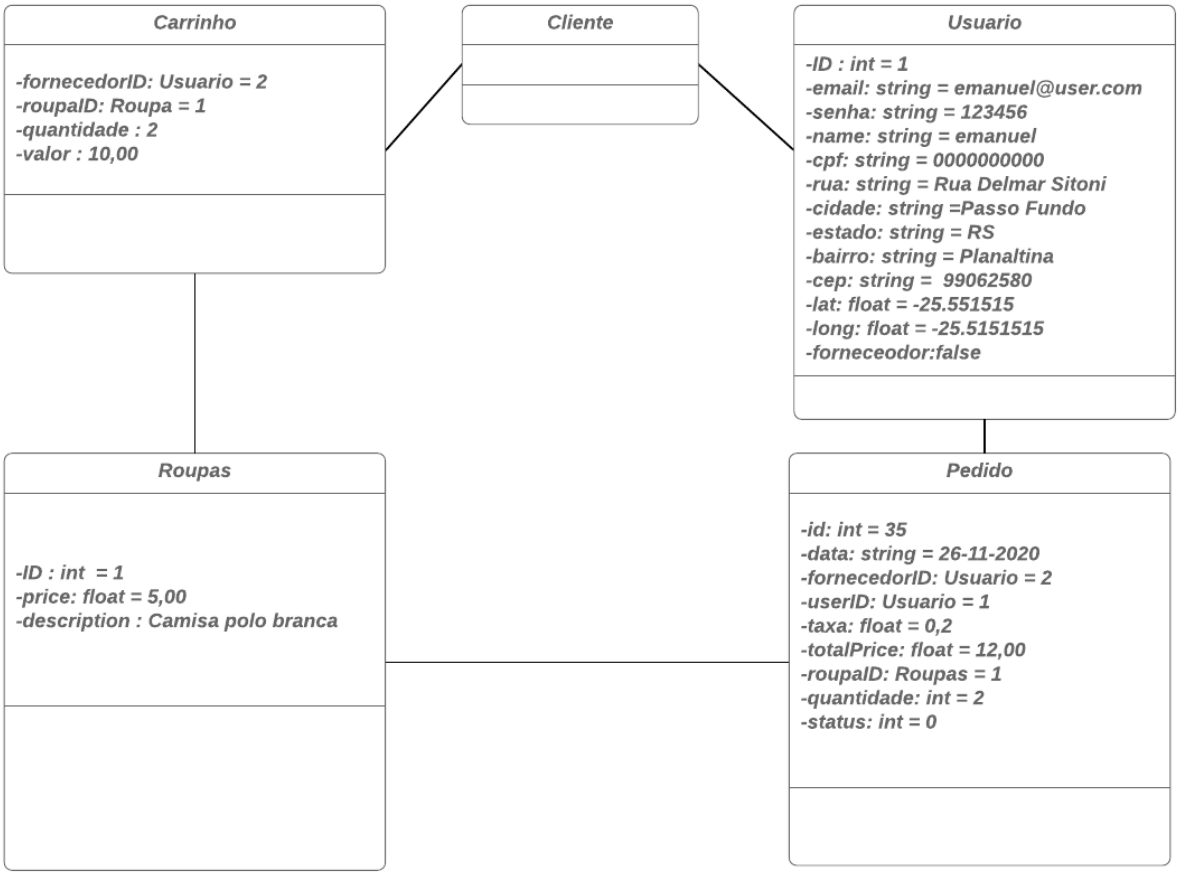


***Fonte**: Do autor, 2020*

3.5 Diagrama de Objetos

O diagrama de objetos da Figura 4, representa somente os dados do usuário do tipo cliente, por conta disso existem as informações do carrinho vinculada a ela, e apartir do carrinho são mostadas as informações do pedido.

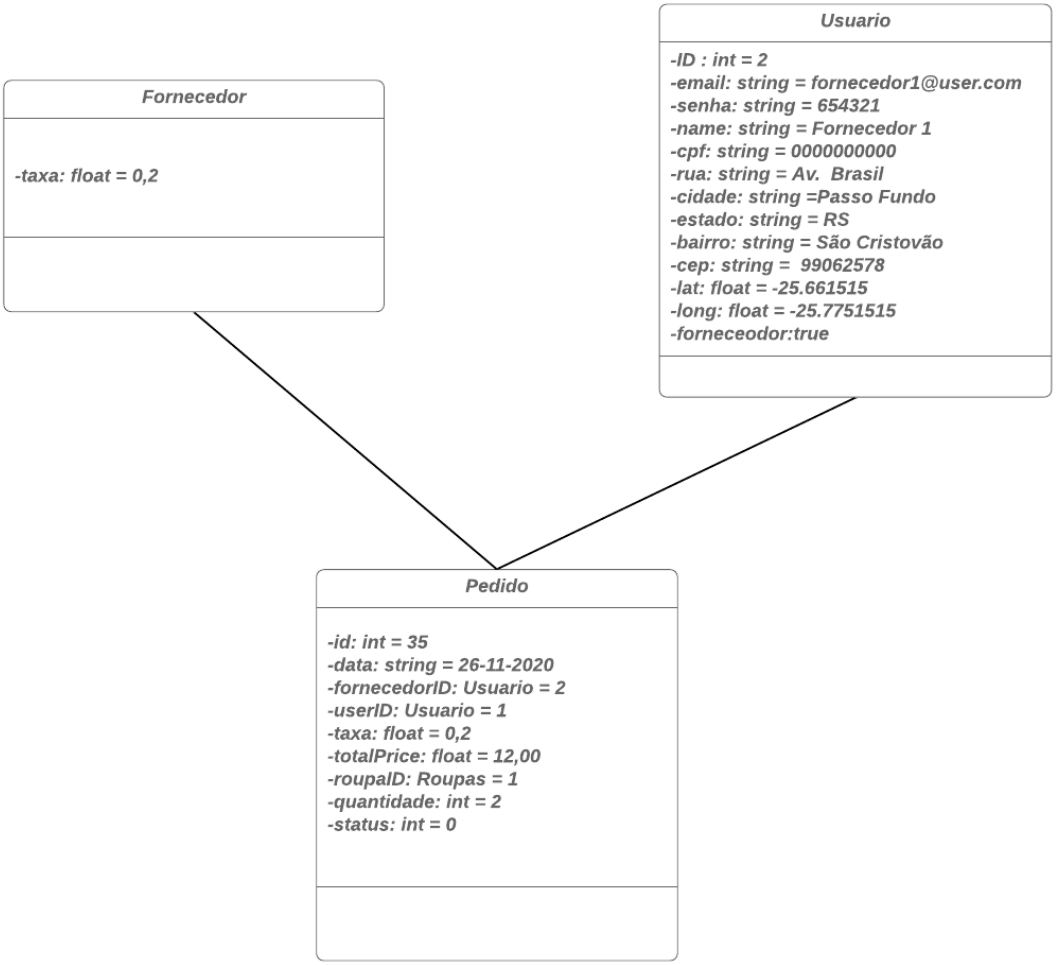
Figura 4 - Diagrama de Objetos (Cliente)



***Fonte**: Do autor, 2020*

Na Figura 5, são exibidos os dados do usuário que é fornecedor do serviço, ele possui apenas vinculo ao pedido, que contém todas as informações necessárias para a realização do serviço.

Figura 5 - Diagrama de Objetos (Fornecedor)



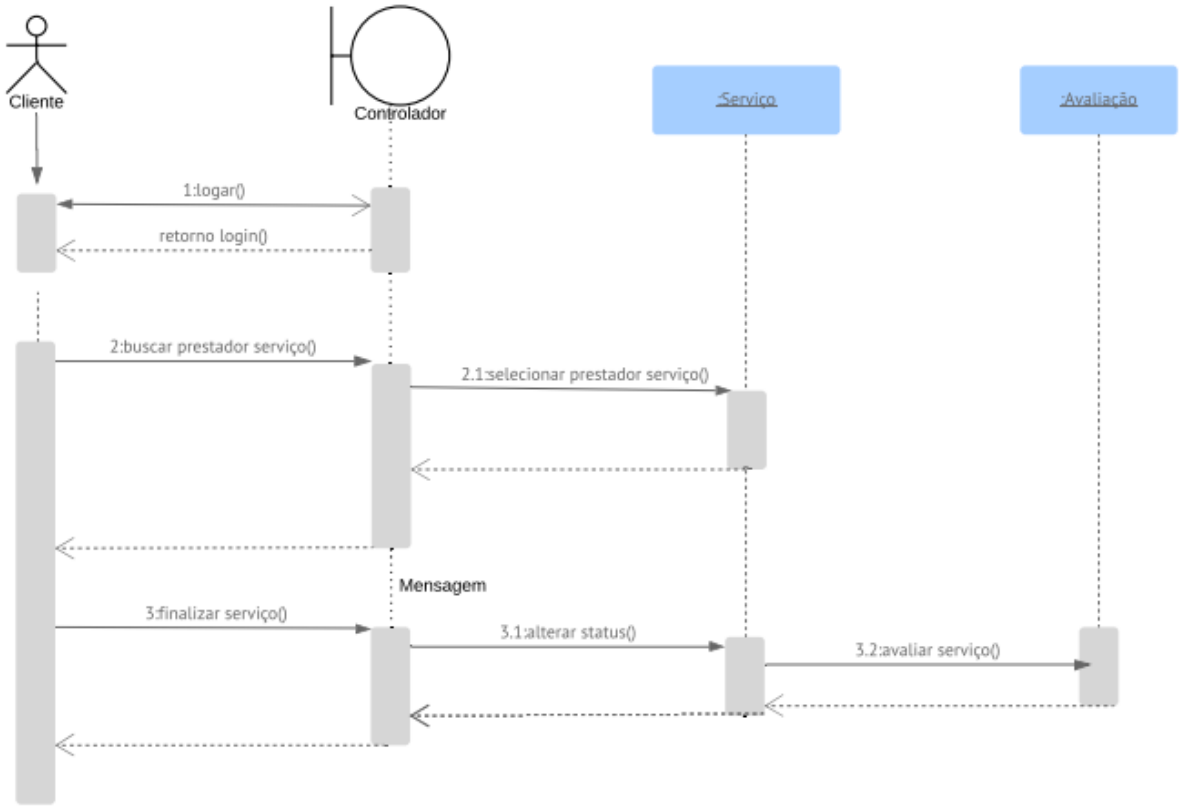
***Fonte**: Do autor, 2020*

3.6 Diagrama de Sequência

O diagrama de sequência se preocupa em demostrar a ordem temporal em que as mensagens são trocadas entre objetos envolvidos em um determinado processo(DEVMEDIA,2009).

No diagrama de sequência da Figura 5 mostra a interação do cliente com o controlador, que por sua vez gerencia a chamada das funções e a comunicação com objetos de serviço e avaliação.

Figura 6 - Diagrama de Sequência



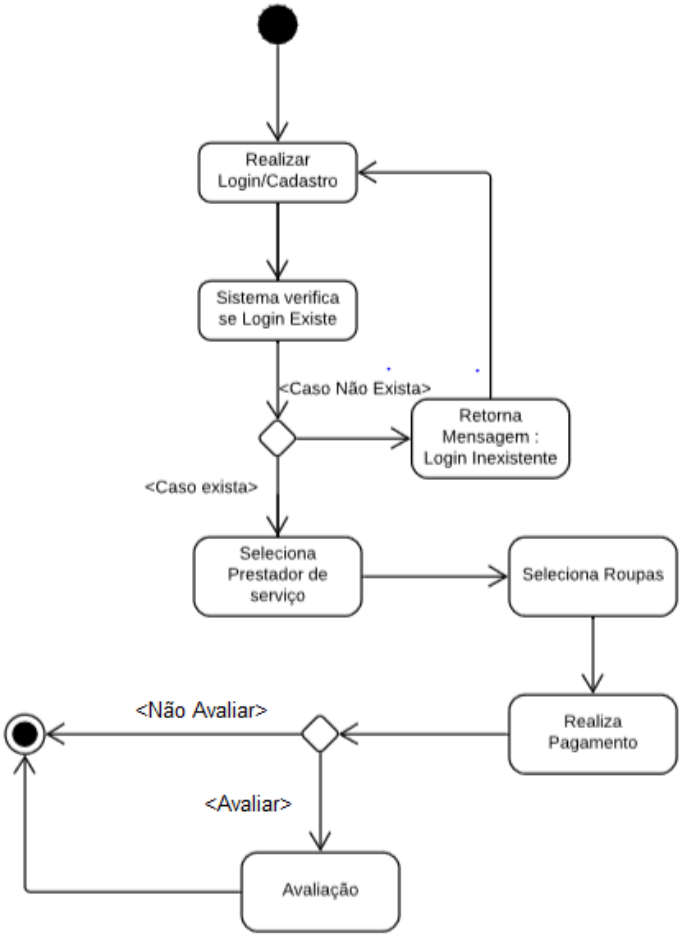
***Fonte**: Do autor, 2020*

3.7 Diagrama de Atividades

O diagrama de atividade apresenta a modelagem de aspectos dinâmicos do sistema e envolve a modelagem de etapas sequenciais.

Na Figura 6 é apresentado o fluxo da atividade da aplicação. Desse modo na primeira interação podemos verificar que o sistema faz a validação se o usuário possui login ou não, caso não possua ele é redirecionado para a tela de login/cadastro novamente. Após selecionar o prestador de serviço, as roupas e realizar o pagamento, o usuário terá a escolha de realizar ou não a avaliação, (a avaliação não é obrigatória), e finalizar a atividade.

Figura 7 - Diagrama de Atividades



***Fonte**: Do autor, 2020*

4 DESENVOLVIMENTO

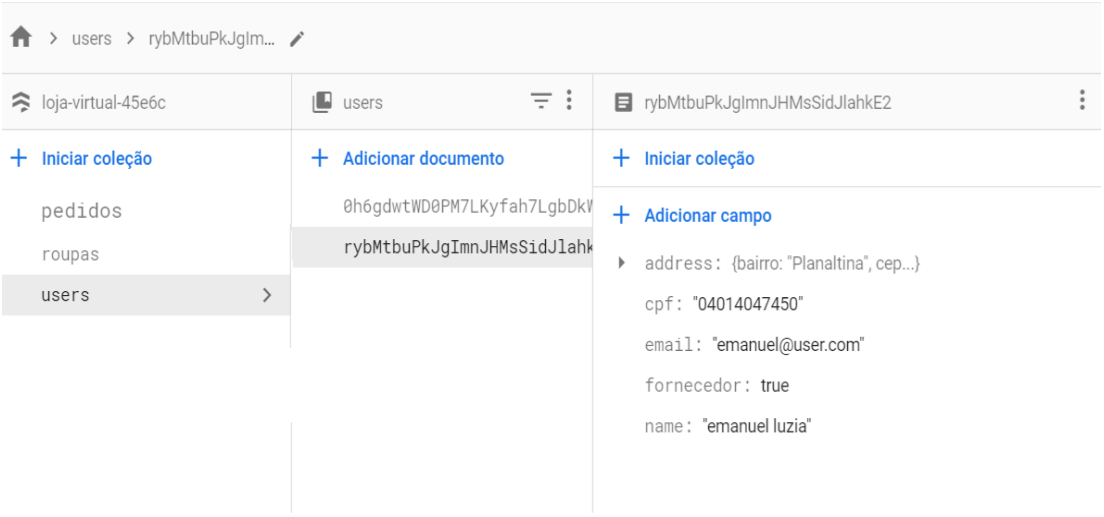
Nesta seção serão apresentadas as ferramentas e recursos utilizados no desenvolvimento da aplicação.

4.1 Banco de dados

O banco de dados utilizado foi o Cloud Firestore, ele é um banco de dados NoSQL, por conta disso toda a sua organização é feita por meio de coleções, subcoleções e documentos, conforme é exibido na Figura 8. O banco de dados foi organizado em 4 coleções:

- **Pedidos:** Coleção armazena as informações dos pedidos, possue o id do cliente e do fornecedor, bem como o valor dos itens e a quantidade.
- **Roupas:** Coleção responsável por armazenar os dados das roupas.
- **Users:** Coleção armazena as informações dos usuários, dentro dessa coleção o usuário é identificado como sendo um fornecedor de serviço ou um cliente, atraves de com campo booleano chamado fornecedor.
    - **Carrinho:** A coleção carrinho fica dentro da coleção Users e armazena as informações do carrinho, que posteriormente vão virar um pedido na coleção Pedidos.

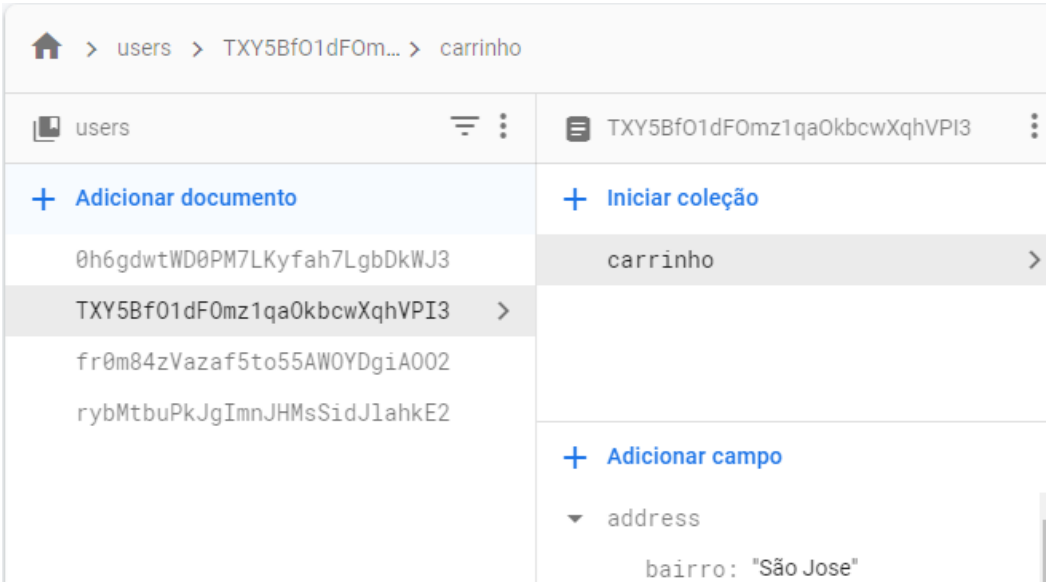
Figura 8 – Coleções Firebase



***Fonte**: Do autor, 2020*

A Figura 9, demonstra como é a estrutura da coleção carrinho, que é criada a dentro da coleção Users, a estrutura foi feita desta forma, pois o carrinho vai ser único para cada usuário do sistema.

Figura 9 - Coleção Carrinho



***Fonte**: Do autor, 2020*

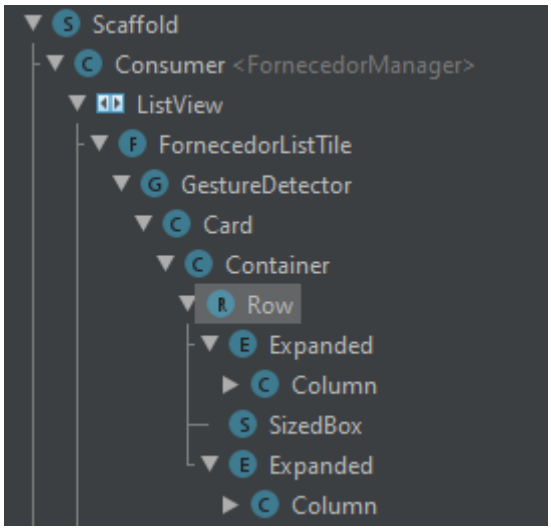
4.2 Autenticação

Para realizar a autenticação dos usuários foi utilizado o recurso do firebase chamado *Authentication*, é por meio dele que é realizado o cadastro dos usuários e o gerenciamento das sessões. As informaçoes necessárias para o Firebase gerar o ID do usuário, são e-mail e senha, depois que firebase retorna o ID do usuário ao aplicativo, é criado um documento na classe users, com o ID do usuário, a onde vai ser gravado então o restante das informações solicitadas no momento do cadastro.

4.3 Interface

No *Flutter* toda a interface é construída através de *widgets*, eles moldam desde o comportamento da tela, até seu layout (botões, fontes, cores, espaçamentos). Eles são organizados em uma hierarquia de árvore, conforme a Figura 10 apresenta de maneira ilustrativa.

Figura 10- Hierarquia Árvore



***Fonte**: Do autor, 2020*

Os *widgets* são separados em dois tipos *Layout* e *interface*, eles também possuem dois estados:

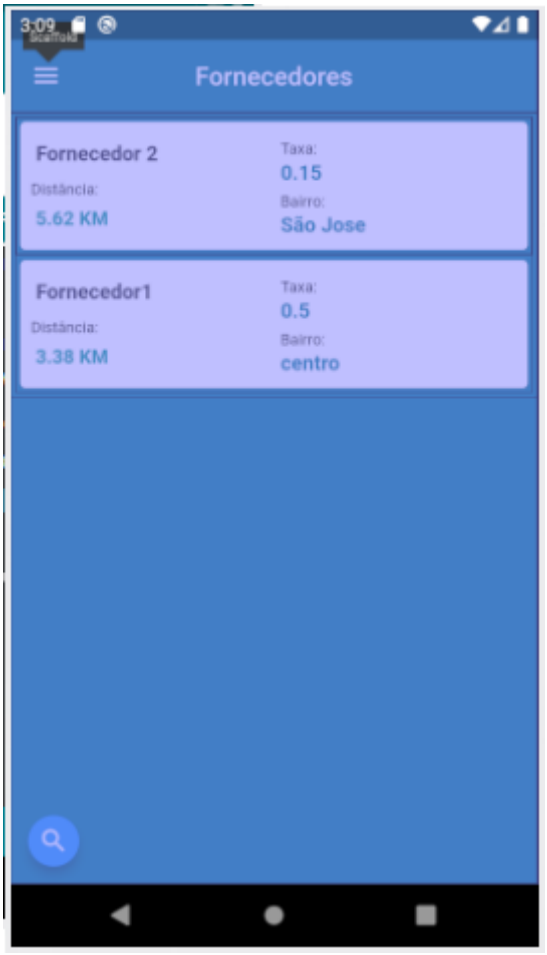
- ***Stateless:*** *É* utilizado para estruturas mais simples, após ser construído e permanece imutável, caso uma informação seja atualizada, é necessário recarregar a tela inteira, normalmente é utilizado em telas mais simples que possuam um conteúdo fixo.
- ***Statefull:*** Consegue guardar o estado da *view*, fazendo com que sempre que ocorra alguma alteração no estado, o *Flutter* consiga observar essa mudança e atualizar o *widgets*, assim atualizando somente os *widgets* necessários.

4.3.1 Widgets Layout

Os *widgets* de layout são utilizados para posicionar outros *widgets*, os mais utilizados nesse trabalho foram:

- **Scaffold:** É a estrutura básica da tela, todos os demais Widgets, ficam dentro dele, na Figura 11, o *Scaffold* foi selecionado, como todos os demais Widgets se originam dele, toda a tela ficou selecionada.

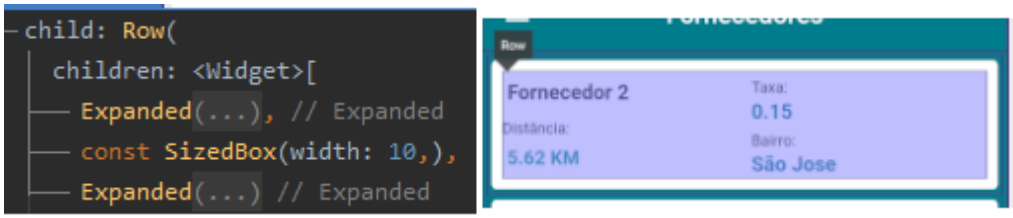
Figura 11 - Scaffold



***Fonte**: Do autor, 2020*

- **Row:** Organiza os *widgets* em uma matriz horizontal, a Figura 12-A demonstra a sintaxe do widgets. Já na Figura 12-B é exibido como todos os filhos da Row são organizados de forma horizontal.

Figura 12 - Row

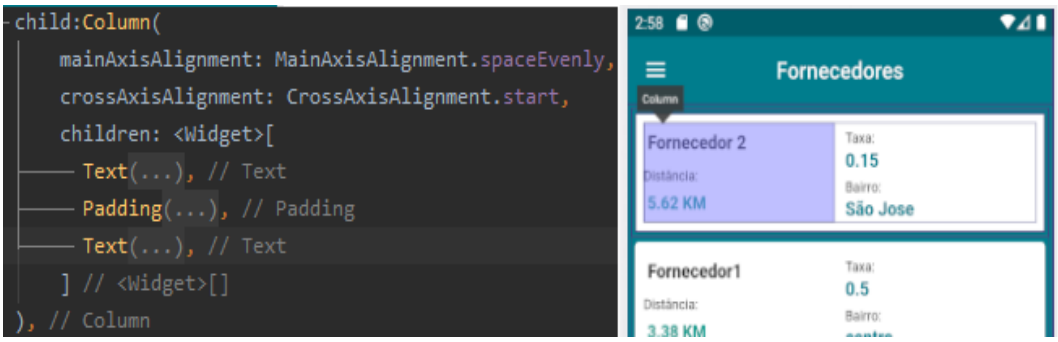


A ***B***

***Fonte**: Do autor, 2020*

- **Column:** Como no nome sugere é utilizado para organizar os *widgets* em colunas, na Figura 13-A é exemplificado como a *Column* é utilizada dentro do código e na Figura 13-B é exibido como se manifesta no *front-end*. Dentro da *Column* existem outros *widgets* como seus filhos, no exemplo utilizado a *Column* é filha da *Row*, que por usa vez é filha do *Scafold*.

Figura 13 - Column



A ***B***

***Fonte**: Do autor, 2020*

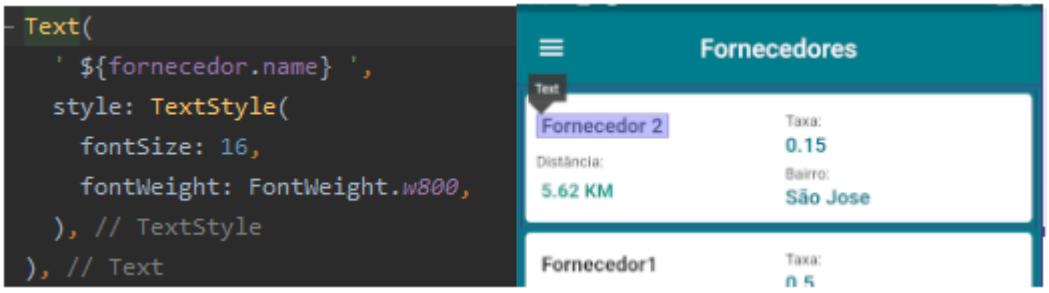
4.3.2 Widgets Interface

Os *widgets* de interface são utilizados para criar componentes visuais, obrigatoriamente eles precisam estar incluídos em um *widget* de layout, podem também ser filhos de outro *widget* de interface, desde que ele esteja em algum nível abaixo de um *widget* de layout, assim eles são posicionados na tela. Alguns exemplos dos Widgets de Interface:

.

- **Text:** Exibe uma string, possui o elemento Style que é utilizado para customizar visualmente o texto. A Figura 14-A exibe exemplo de como o *Text* é utilizado no código e a Figura 14-B demonstra como ele é exibido no *front-end*.

Figura 14 - Text

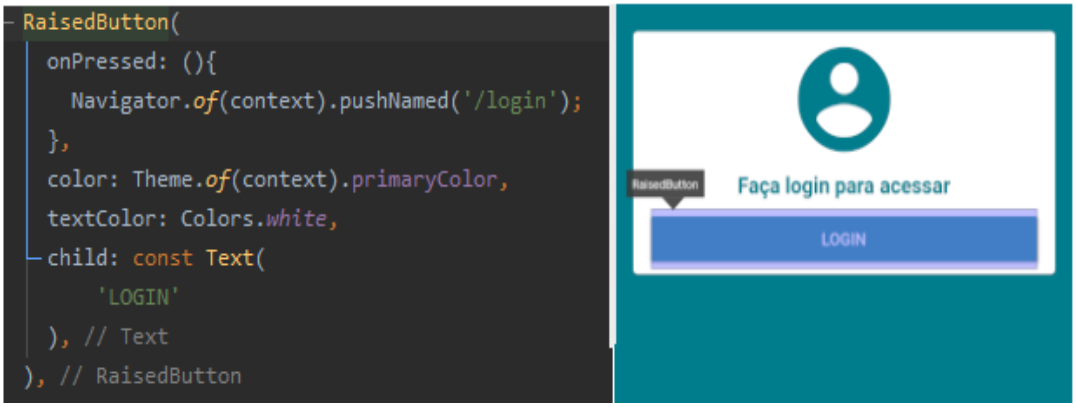


A ***B***

Fonte: *Do autor, 2020*

- **RaisedButton:** É um botão, possui dentro dele o elemento onPressed, que executa alguma função quando o botão é pressionado, na Figura 15-A é demonstrado como o *RaisedButton* é usado no código. No exemplo da Figura 15-B exibe o botão no *front-end*, quando o botão é pressionado ele está redirecionando o usuário para a tela login. Vale ressaltar que nesse exemplo o RaisedButton possui um *Widget* de texto como filho.

Figura 15 - RaisedButton

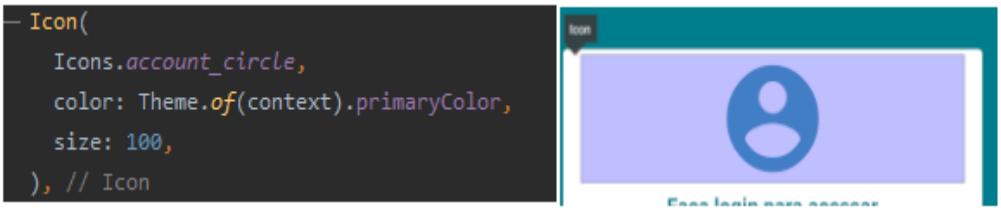


A ***B***

Fonte: *Do autor, 2020*

- **Icon:** É utilizado para exibir os ícones disponíveis no Flutter, possui também alguns elementos de customização. A Figura 16-A, exibe um exemplo de sua implementação no código, e a Figura 16-B, exibe o resultado no *front-end*.

Figura 16 - Icon



A ***B***

Fonte: *Do autor, 2020*

4.4 Gerência de estado

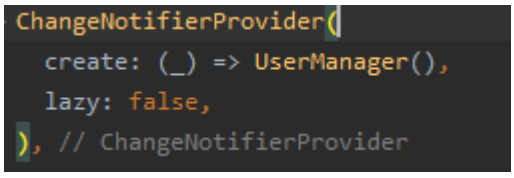
Para auxiliar na gerencia de estado, foi utilizada um pacote externo chamado *Provider*, ele consegue carregar um objeto em memória, e assim esse objeto consegue ser reutilizado e acessado de qualquer local do aplicativo.

Para realizar o carregamento dos objetos em memória, foram utilizados dois tipos de *Providers* dentro do main.dart (arquivo que carrega toda vez que a aplicação é compilada).

4.4.1 ChangeNotifierProvider

É responsável por carregar o objeto em memória, Na Figura 17, o elemento chamado “create” está carregando em memória os dados da classe *UserManager*, que possui as informações do usuário.

Figura 17 - ChangeNotifierProvider



Fonte: *Do autor, 2020*

Existe também o elemento chamado *lazy*, que no exemplo da Figura 17, esta marcado como false, por padrão *lazy* vem como true, ele faz com que o objeto apenas seja carregado quando alguma tela que consuma a classe *UserManager*

seja acessada, quando *lazy* está marcado como false, o carregamento em memória ocorre quando o arquivo main.dart é executado.

4.4.2 ChangeNotifierProxyProvider

É semelhante ao ChangeNotifierProvider, porém possui mais responsabilidades, ele recebe duas classes, uma ele irá carregar em memória, e a outra através do elemento “update”, ele irá monitorar, e caso ocorra alguma mudança na classe monitorada, irá ocorrer a execução da função informada.

Na Figura 18 é exibido o *provider* responsável por carregar a classe FornecedorManager, ele está monitorando a classe *UserManger*. Esse *provider* está fazendo isso pois quando o usuário realiza *login*, é calculado a distância entre o usuário e fornecedor de serviço.

Essa distância é salva dentro dos dados da classe FornecedorManger, caso o usuário seja alterado, é necessário recarregar os dados dos fornecedores para que a distância do novo usuário logado seja calculada, esse *provider* está fazendo exatamente isso, quando o usuário é alterado, a função updateUser, da classe FornecedorManager, é executada, e a distância entre os fornecedores e o novo usuário logado, é calculada.

Figura 18 - ChangeNotifierProxyProvider

```
ChangeNotifierProxyProvider<UserManager, FornecedorManager>(
  create: (_) => FornecedorManager(),
  lazy: false,
  update: (_, userManager, fornecedorManager) =>
  fornecedorManager..updateUser(userManager.user),

), // ChangeNotifierProxyProvider
```

***Fonte**: Do autor, 2020*

4.4.3 Consumer

O objeto carregado em memória pode ser acessado através do método *Consumer*. É possível também exibir os dados acessados no *front-end*, no exemplo da Figura 19, o *Consumer* está observando a classe CarrinhoRoupa e acessando a quantidade de itens.

Figura 19 – Método Consumer

```
Consumer<CarrinhoRoupa>(
    builder: (_, carrinhoRoupa, __){
    return Column(
      children: <Widget>[
        CustomIconButton(
          iconData: Icons.add,
          color: Theme.of(context).primaryColor,
          onTap: carrinhoRoupa.increment,
        ), // CustomIconButton
        Text(
          '${carrinhoRoupa.quantidade}',
          style: const TextStyle(fontSize: 20),
        ), // Text
        CustomIconButton(
          iconData: Icons.remove,
          color: carrinhoRoupa.quantidade > 1 ? Theme.of(context).primaryColor : Colors.red,
          onTap: carrinhoRoupa.decrement,
        ), // CustomIconButton
      ], // <Widget>[]
    ); // Column
  },
) // Consumer
```

***Fonte**: Do autor, 2020*

Caso essa informação seja alterada, o próprio *Consumer* vai recarregar apenas os widgets necessários, assim evitando que toda a tela seja recarregada. Dessa forma o Consumer delega para si, as funções que originalmente seriam desempenhadas pelos estados dos widgets (Stateless, Statefull).

4.4.4 NotifyListener

Outro método que foi amplamente utilizado foi o *notifyListener*, ele indica quando ocorreu alguma alteração na classe, no exemplo da Figura 20, é exibida uma função que remove os itens do carrinho. Como o método *notifyListener* está sendo chamado, assim que o item é removido, todas as telas que estão observando essa classe através do *Consumer*, vão ter as informações atualizadas.

Figura 20 - Método notifyListeners

```
void removefromCarrinho(CarrinhoRoupa carrinhoRoupa){
  items.removeWhere((r) => r.id == carrinhoRoupa.id);
  user.carrinhoReference.document(carrinhoRoupa.id).delete();
  carrinhoRoupa.removeListener(_onItemUpdated);
  notifyListeners();
}
```

***Fonte**: Do autor, 2020*

No momento em que a função for executada, a quantidade do carrinho será alterada, e o valor que está sendo exibido conforme mostra a Figura 19, é alterado.

4.5 Pacotes Externos

Foram utilizados pacotes externos para o desenvolvimento da aplicação, alguns para a utilização do Firebase e outros para facilitar o desenvolvimento, são esses pacotes:

- **Provider:** Utilizado para auxiliar na gerencia de estado da aplicação e dos Widgets (PROVIDER, 2020);
- **Dio:** Cliente HTTP para a linguagem Dart;
- **Brasil_fields:** Utilizado para formatar campos que só existem no Brasil, como CPF ou CEP;
- **Cloud_firestore:** Permite a utilização do Cloud Firestore API,é reponsavel por possibilitar o CRUD das informações no banco de dados do Firebase;
- **firebase_auth:** Permite a utilização do sistema de autenticação do Firebase;
- **CEPAberto:** Está sendo utilizada a API do CEPAberto, ele disponibiliza gratuitamente códigos de Endereçamento Postal (CEP) geolocalizados em todo o Brasil (API CEP ABERTO, 2020). A API possui algumas limitações, como o limite de 10 mil requisições por dia, e também o intervalo de 1 segundo entre as requisições. A API é utilizada para salvar as informações do endereço do usuário. Com a informação disponibilizada da latitude e longitude é possível realizar o cálculo da distância entre os usuários, dessa forma exibir a distância entre o cliente e os fornecedores de serviços;

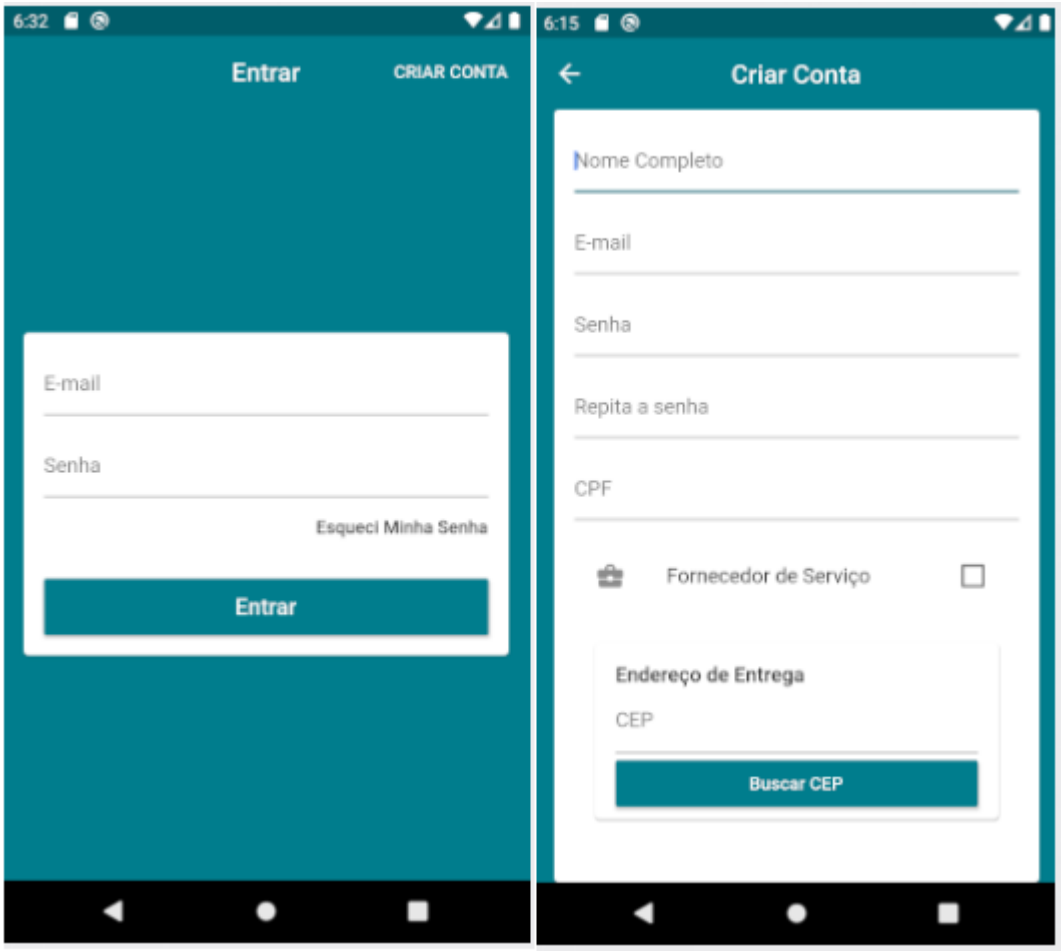
- **Geolocator:** Pacote que permite realizar cálculos de distância por meio da latitude e longitude.

5 RESULTADOS

Nesta seção seram apresentados os resultados construídos utilizando as tecnologias e conceitos propostos.

A tela inicial do sistema é da de login, no canto superior direito existe o botão “CRIAR CONTA”, para o usuário realizar seu cadastro caso ainda não possua. Na tela de cadastro são solicitadas as informações necessárias, conforme é exibido na Figura 21.

Figura 21 - Tela Cadastro/Cadastro

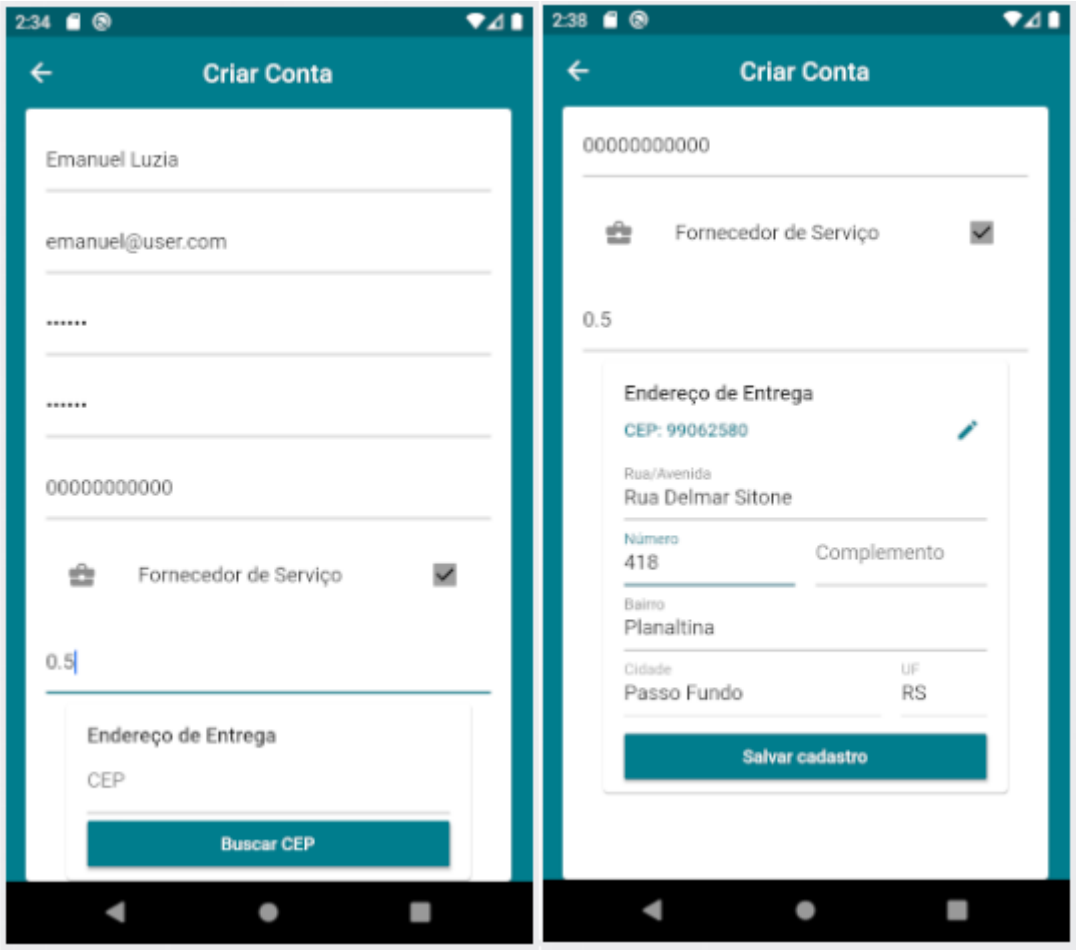


***Fonte**: Do autor, 2020*

Existe um checkbox chamado ‘Fornecedor de Serviço’, ele tem o intuito de indicar se esse usuário vai ser um fornecedor de serviço ou um cliente, caso o checkbox esteja marcado como TRUE, o usuário será um fornecedor de serviço, e será habilitado um campo para informar a taxa com que o fornecedor vai trabalhar,

caso checkbox não seja marcado, o usuário será o cliente, a única informação que difere os dois além do tipo, é o campo de taxa, que é exclusivo do fornecedor. Conforme é exibido na Figura 22, existe também a informação do CEP, é utilizada a API do CEPaberto para buscar as informações do endereço, baseado no CEP informado, ao clicar no botão de “Buscar CEP”, é feita a requisição para a API que retorna os dados da Rua, bairro, cidade, estado, latitude e longitude. São exibidos também os campos de número e complemento, para que o usuário complete o cadastro.

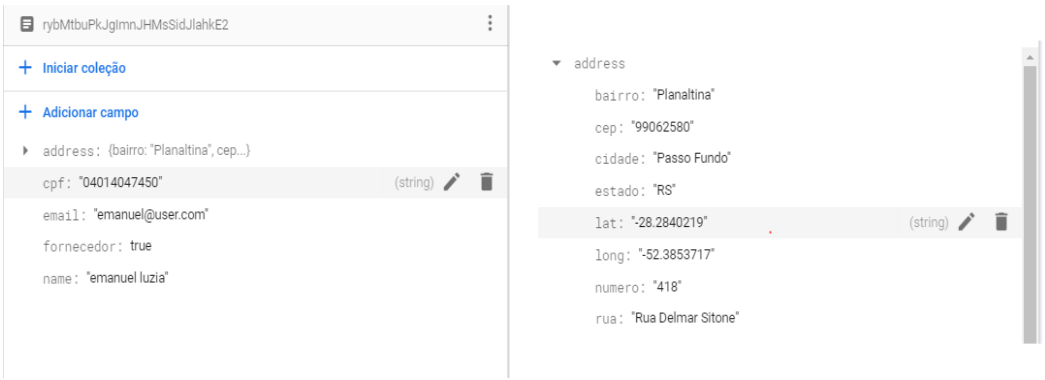
Figura 22 - Tela Cadastro Endereço



***Fonte**: Do autor, 2020*

No momento em que o cadastro é salvo, é criado um documento na coleção users (Figura 23), com as informações preenchidas.

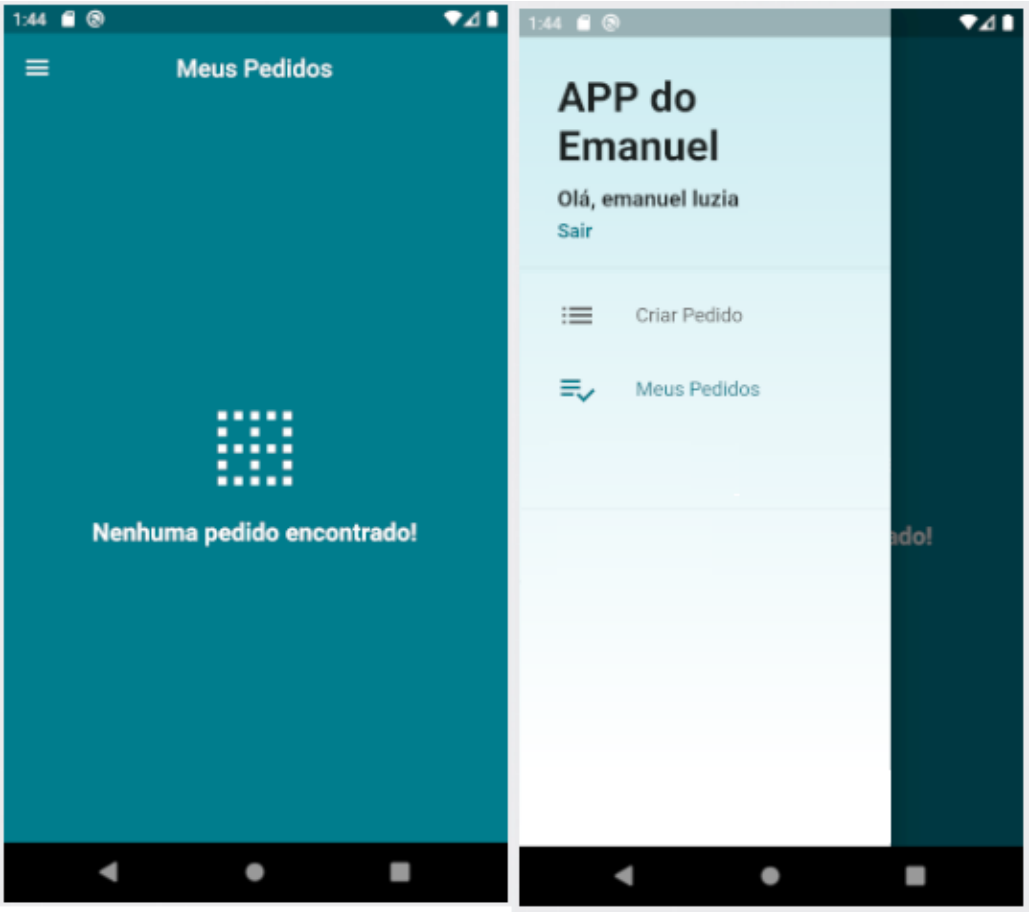
Figura 23 - Coleção Users



***Fonte**: Do autor, 2020*

Após a criação da conta o sistema realiza o *login* e em seguida mostra a tela inicial (Figura 24). A tela inicial do Cliente é a de “Meus Pedidos”, a onde são listados os pedidos do usuário, no menu existem as opções de acesso a tela de “Meus Pedidos” e também a tela de “Criar Pedido”.

Figura 24 - Tela Inicial - Cliente



***Fonte**: Do autor, 2020*

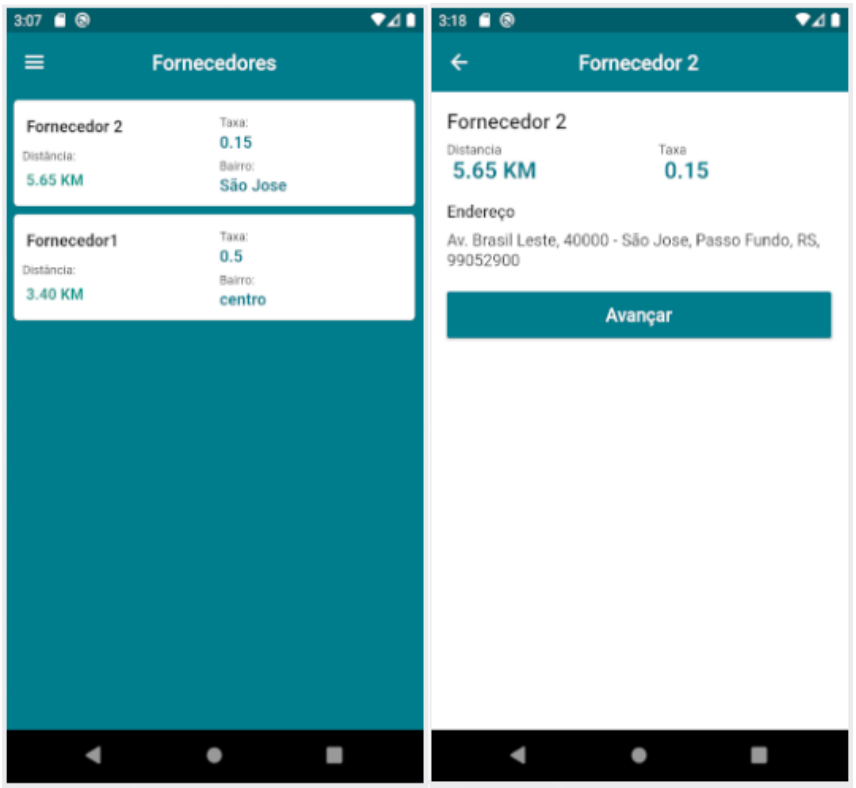
Na tela de “Criar Pedido”, - são listados todos os fornecedores ao cliente, conforme Figura 25. Das informações exibidas do fornecedor, vale destacar a informação da distância e da taxa.

A informação da distância representa a distância que o cliente logado, está dos fornecedores que estão sendo exibidos na listagem, esse cálculo é realizado através do pacote externo *Geolocator*. Ele recebe a latitude e longitude do cliente e do fornecedor e retorna a distância em metros. A informação da distância é convertida em quilômetros para ser exibida ao usuário.

A informação da taxa vem do cadastro do fornecedor, essa taxa é aplicada individualmente em cada roupa no momento da criação do pedido.

As roupas cadastradas possuem um valor fixo definido pelo sistema, por conta disso a relação da taxa e a distância é justamente o que vai causar a concorrência no momento em que o cliente escolher o fornecedor. Ao Clicar em um dos fornecedores é a aberta a tela que exibe os detalhes do fornecedor, conforme é exibido na Figura 25.

Figura 25 - Listagem Fornecedor - Cliente

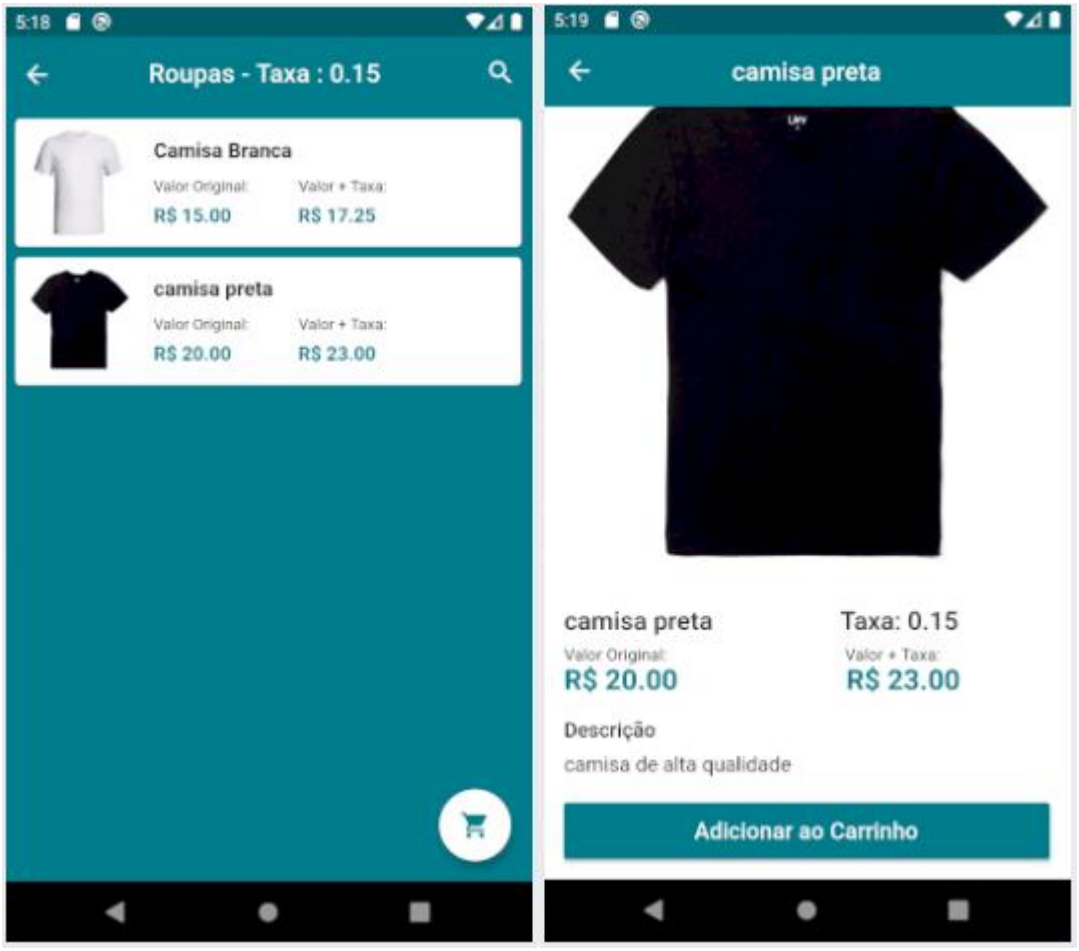


***Fonte**: Do autor, 2020*

Quando o cliente avançar, será aberta a listagem das roupas, exibindo o valor original da peça de roupa e também valor da peça de roupa já com a taxa aplicada.

No momento em que o cliente clica nas roupas, é aberto a tela de detalhes, a onde são exibidas as informações daquela roupa e também é exibido botão que irá adicionar a peça de roupa ao carrinho, esse processo é exibido na Figura 26.

Figura 26 - Seleção de Roupas - Cliente



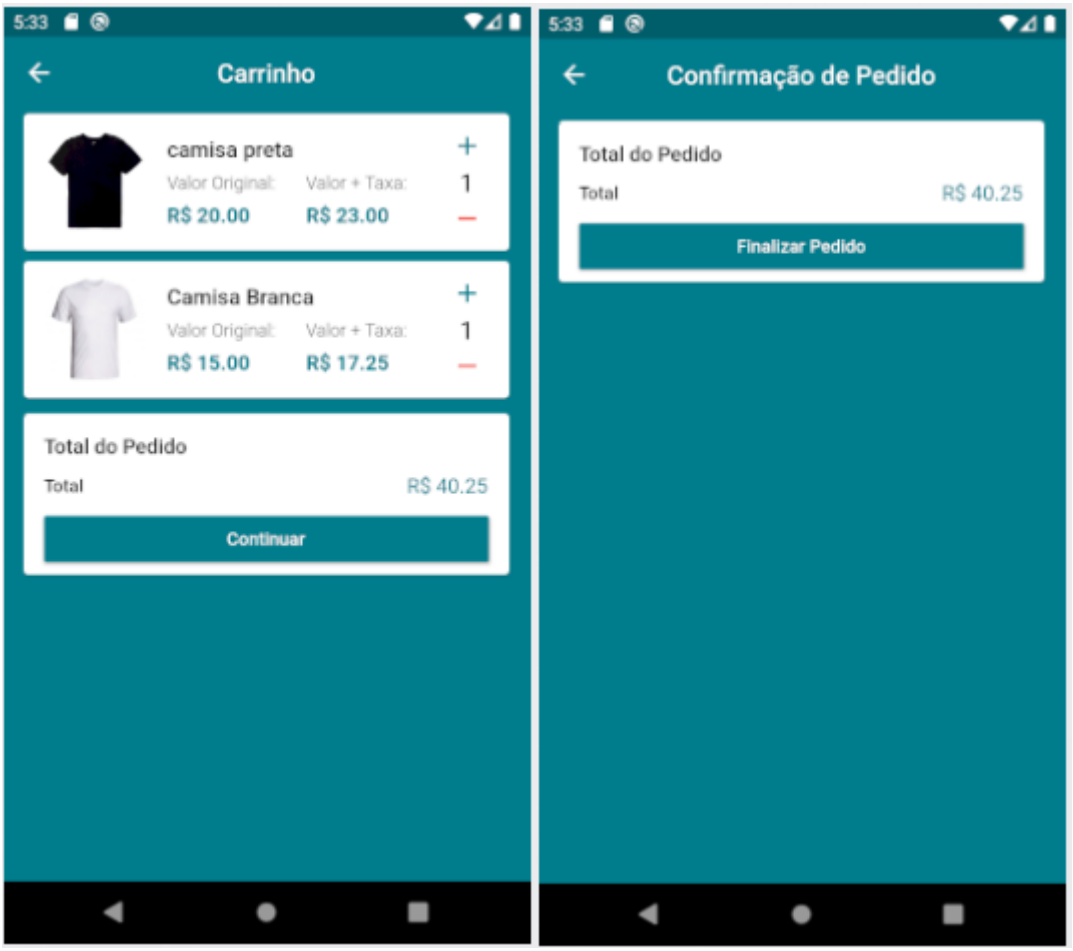
***Fonte**: Do autor, 2020*

Quando usuário adiciona o produto ao carrinho, é criada a coleção do carrinho no *firebase*, essa coleção contém as informações do ID do fornecedor, a quantidade e o ID da peça de roupa, após criação da coleção ocorre o redirecionamento para a tela do carrinho.

Na tela do carrinho é possível adicionar e remover a quantidade de itens, e também é exibido o total do pedido, conforme a Figura 27 exibe. Ao clicar em

continuar será exibido total do pedido, nesse momento seria aonde o usuário cadastraria o cartão de crédito, para posteriormente, quando o pedido for concluído, possa ocorrer o pagamento do serviço, porém a tela aonde ocorreria o cadastro do cartão não foi criada, pois foi priorizado a implementação das funções básica do aplicativo.

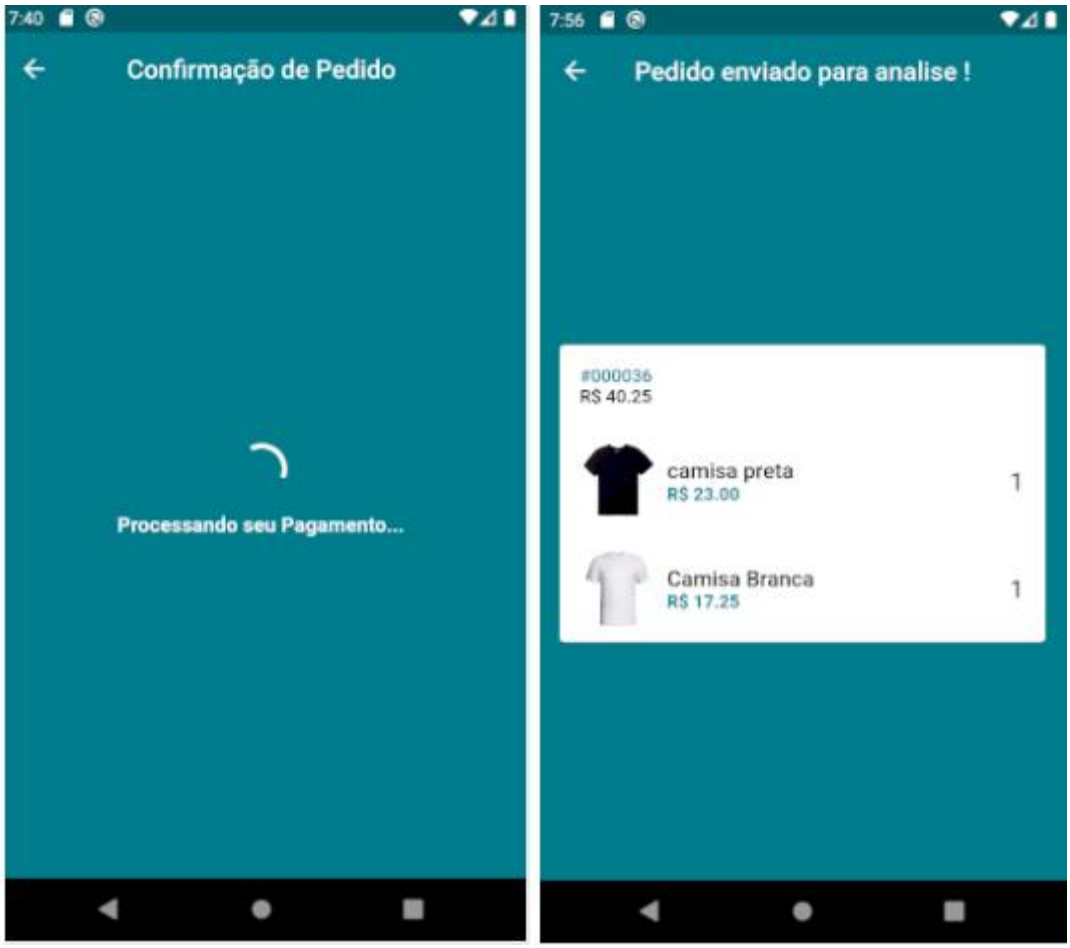
Figura 27 - Tela Carrinho - Cliente



***Fonte**: Do autor, 2020*

Quando o pedido é finalizado é o momento aonde ocorre a criação do documento na classe pedidos no *firebase*, essa classe contém um ID único e sequencial, e possui as informações do pedido, como os itens, o valor de cada um deles, quantidade, e também o ID do fornecedor e do cliente que foram vinculados ao pedido. Para confirmar que o pedido foi criado com sucesso é exibido um resumo ao usuário, conforme é exibido na Figura 28.

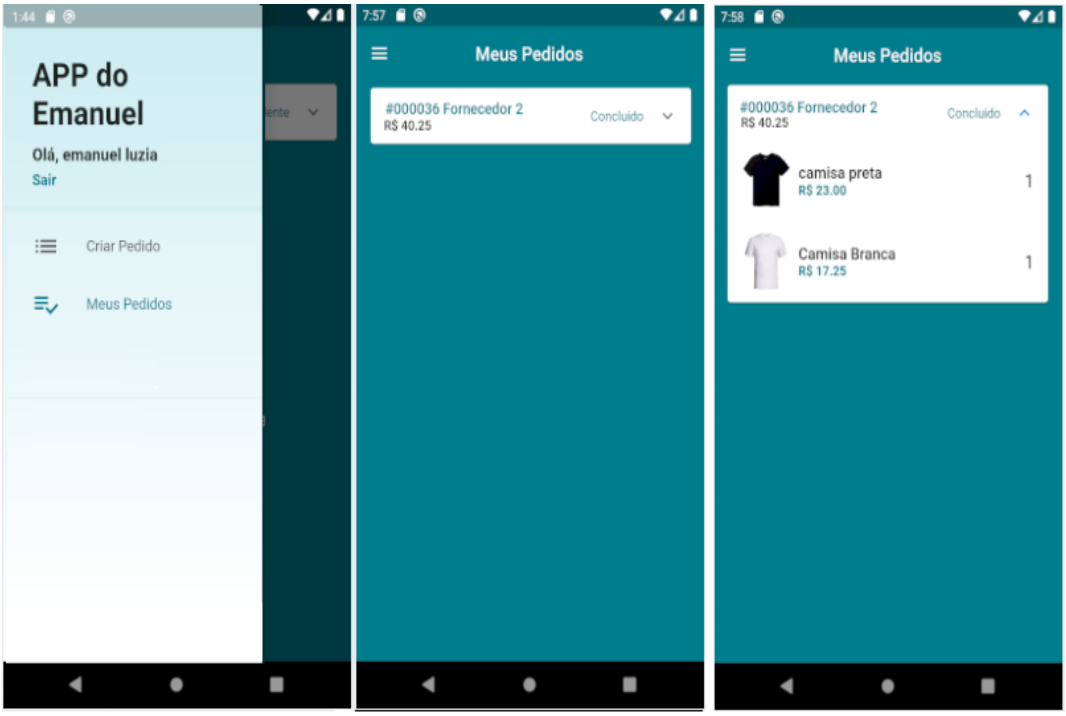
Figura 28 - Confirmação do Pedido - Cliente



***Fonte**: Do autor*

Após a criação dos pedidos eles são listados na tela de “Meus pedidos”, nessa tela é exibido um resumo que possui o ID do pedido e do fornecedor, e também os itens, valor total do pedido e o Status no qual o pedido se encontra, conforme é exibido na Figura 29.

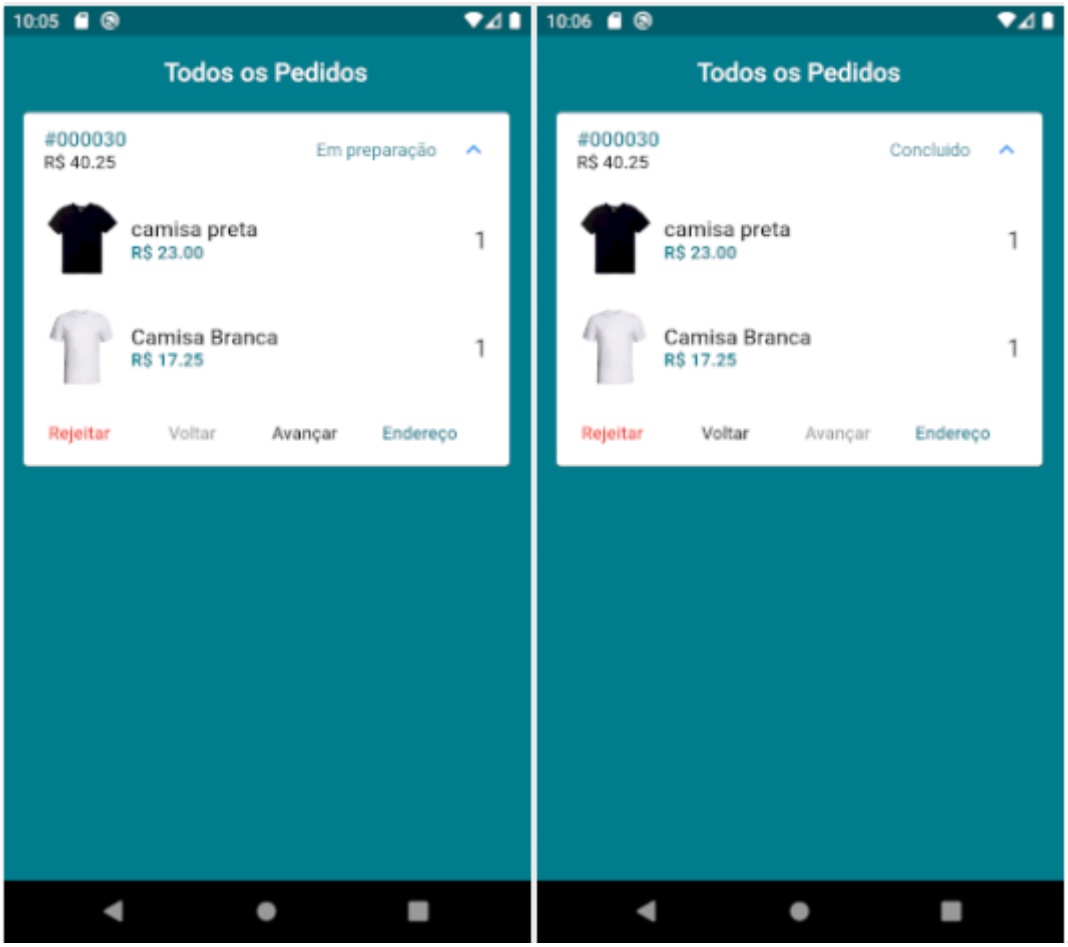
Figura 29 - Consulta do Pedido - Cliente



***Fonte**: Do autor, 2020*

Após pedido criado, o cliente precisa esperar o fornecedor aceitar o pedido para dar continuidade ao processo, o pedido será exibido ao fornecedor para aprovação, na tela de “Todos os Pedidos”, ela é a única tela que o usuário de tipo fornecedor possui acesso, nessa tela são mostrados todos os pedidos que estão vinculados ao fornecedor, conforme é exibido na Figura 30.

Figura 30 – Todos os Pedidos - Fornecedor



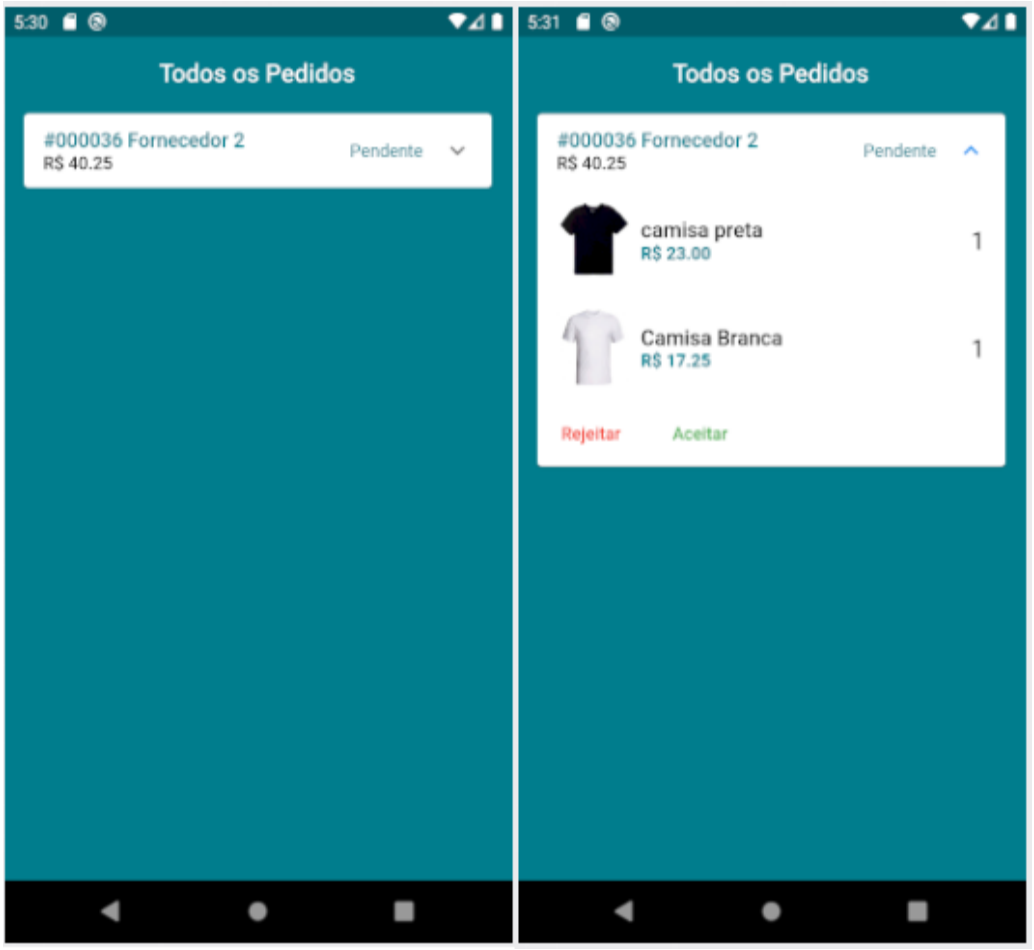
***Fonte**: Do autor, 2020*

Existem 5 Status para o pedido, são eles:

- **Pendente:** É o status padrão para todo pedido aberto, só vai sair deste status quando o fornecedor, aceitar ou recusar o pedido;
- **Rejeitado:** Indica que o Fornecedor do serviço não aceitou o pedido, e o mesmo fica sem a possibilidade de mais interações, também é utilizado quando o cliente cancela o serviço;
- **Em Transporte:** Momento quando o pedido é aceito pelo fornecedor, e o usuário está transportando as roupas ate a casa do Fornecedor;
- **Em preparação:** Ocorre quando o usuário entrega as peças de roupas para o fornecedor, e indica que o serviço está sendo realizador;
- **Concluído:** No momento em que o fornecedor finaliza o serviço, ele indica que o pedido foi concluído.

No momento em que o pedido é criado ele possui o status de Pendente, nesse momento o pedido será listado ao fornecedor para ele aceitá-lo ou rejeitá-lo. Quando o fornecedor realiza login ele só possui a tela de “Todos os Pedidos”, onde vão ser listados os pedidos vinculados a ele, conforme é exibido na Figura 31.

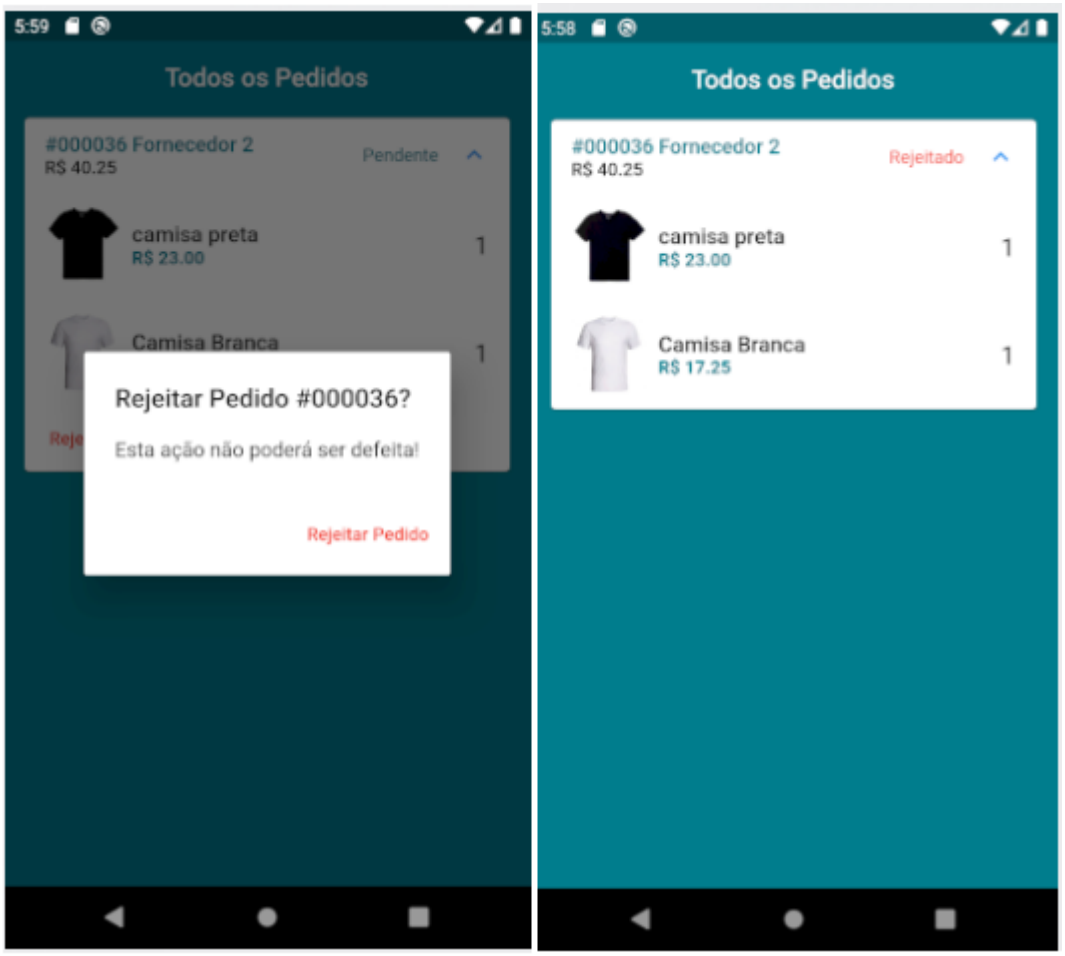
Figura 31 - Aceitar/Rejeitar Pedido - Fornecedor



***Fonte**: Do autor, 2020*

Conforme apresentado na Figura 32, existem duas opções ao fornecedor do serviço, de aceitar ou rejeitar o pedido, caso a opção de rejeitar seja escolhida, abrirá uma caixa de confirmação, se o usuário rejeitar o pedido, ele ficaria com o status de rejeitado, e não haverá mais possibilidade de alteração de status.

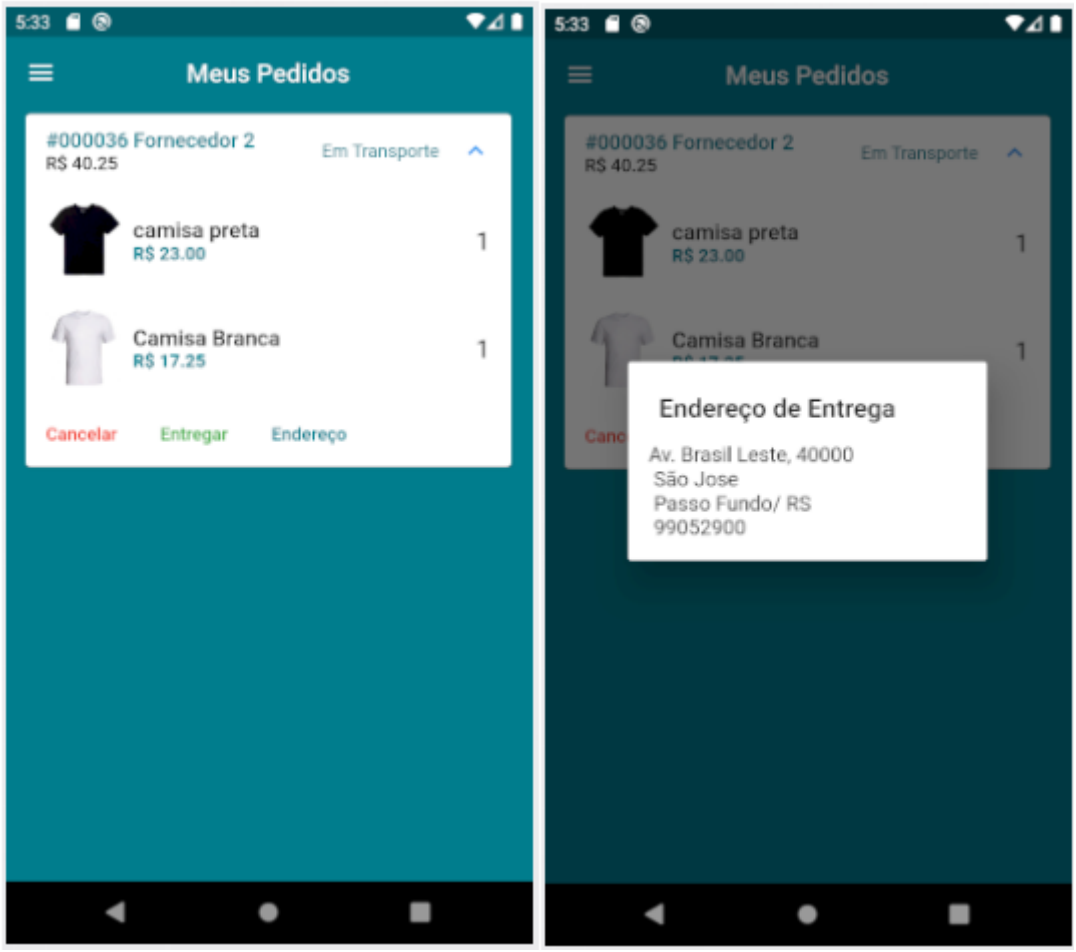
Figura 32 - Rejeitar Pedido - Fornecedor



***Fonte**: Do autor, 2020*

Caso o pedido seja aceito, o status vai ser alterado para “Em Transporte”, isso significa que o cliente está levando as peças de roupas até o fornecedor, por conta disso quando o status for “Em Transporte”, o fornecedor do serviço não poderá realizar nenhuma alteração nos status do pedido, a alteração de status estará disponível ao cliente, que vai informar o momento em que as roupas foram entregues ao fornecedor, é exibido também ao cliente o Botão Endereço, para que ele visualize o endereço do fornecedor, conforme é exibido na Figura 33.

Figura 33 - Entregar Roupas - Cliente

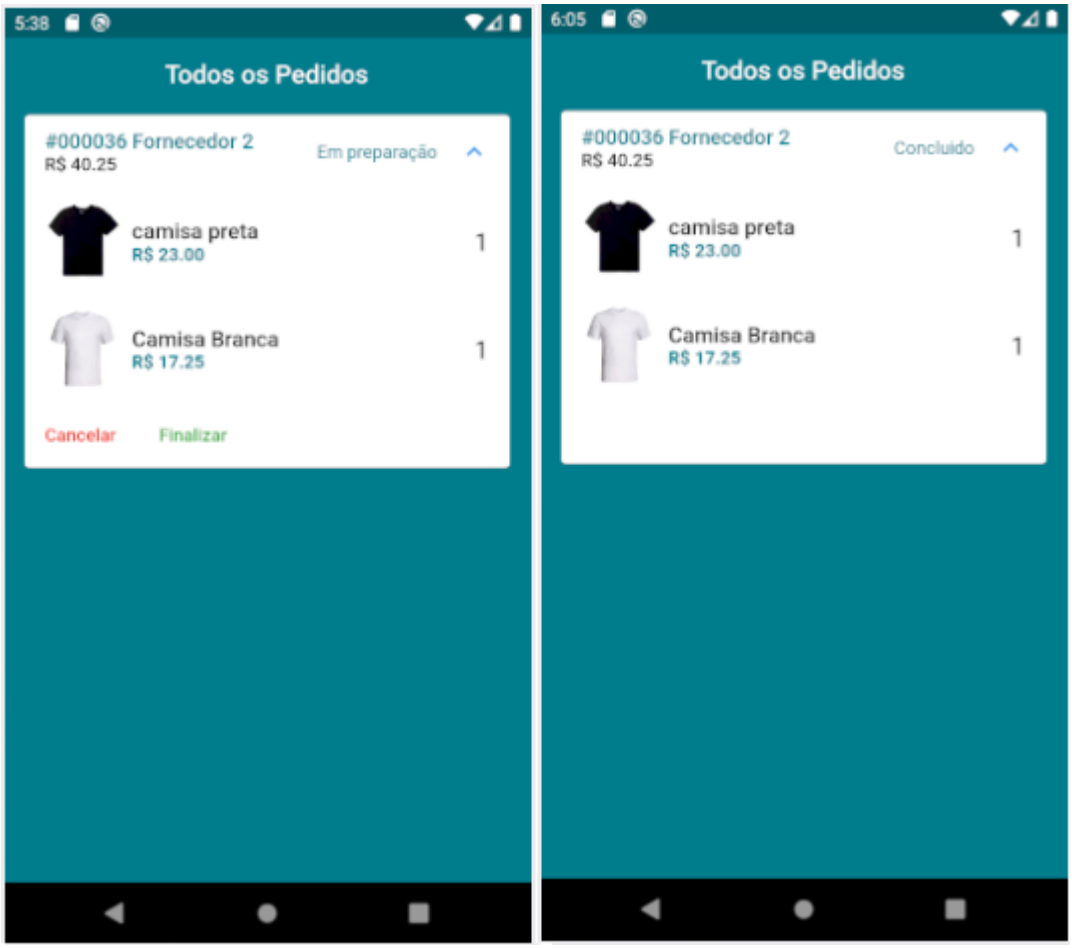


***Fonte**: Do autor, 2020*

No momento em que o cliente informa que entregou as peças de roupa, o status será alterado para” Em Preparação”, sinalizando assim que o serviço de lavagem de roupa está sendo realizado.

Ficará disponível ao fornecedor que informe quando o serviço foi finalizado, conforme é demostrado na Figura 34, quando isso for realizado o status será alterado para concluído, e não haverá mais opções de interação, e assim o processo do pedido é encerrado.

Figura 34 - Finalizar Pedido - Fornecedor



***Fonte**: Do autor, 2020*

6 CONSIDERAÇÕES FINAIS

Neste trabalho foi apresentado o *LaundryApp*, um aplicativo multiplataforma de troca de serviços de lavagem de roupa, que segue os preceitos estabelecidos pela economia compartilhada.

O desenvolvimento do sistema foi feito através de novas tecnologias de desenvolvimento como o Flutter, que permitiu tornar o aplicativo disponível para as duas plataformas móveis predominantes no mercado. E também o *Firebase*, que permitiu uma grande agilidade durante o desenvolvimento devido a sua simplicidade de sua utilização.

No decorrer do trabalho a maior dificuldade encontrada, foi a curva de aprendizado em relação a estrutura do *Front-End* do *Flutter*, que se diferencia do padrão das demais tecnologias previamente conhecidas, isso demandou muita pesquisa, o que acabou prejudicando a quantidade de funcionalidades implementadas.

Para trabalhos futuros podem ser implementar diversas funcionalidades básicas e avançadas que não conseguiram ser implementadas a tempo, como Login com Facebook, criação e edição do perfil dos usuários, bem como possibilitar o cadastro das peças de roupa pelo usuário e a implementação de tipos de frete, que hoje é restrita apenas a uma modalidade, também a utilização de APIs, como de pagamentos, ou do google Maps por exemplo.

7 REFERÊNCIAS

ANEL. Informações do Setor. 2018. Disponível em:
<https://anel.com.br/legislacao-do-setor/>. Acesso em: 09 Nov. 2019.

API CEP ABERTO. Disponível em:
< https://cepaberto.com/>. Acesso em: 10 Nov. 2020.

Arruda, Monique de Souza. INDÚSTRIA 4.0 E MEIO AMBIENTE DO TRABALHO: O DIREITO À SAÚDE NO CONTEXTO DAS INOVAÇÕES DISRUPTIVAS E DA ECONOMIA COMPARTILHADA. Brasil: Amazon Servicos de Varejo do Brasil Ltda, 2019.

FIREBASE. Firebase: 2019. Documentação. Disponível em:< <https://firebase.google.com/?hl=pt-BR>. Acesso em: 17 de out. de 2019.

FLUTTER. Flutter: 2019. Documentação. Disponível em: <https://flutter.dev/docs>. Acesso em: 18 de out. de 2019

Freitas, Gilson. ARDUINO E FIREBASE: Crie seu próprio Dashboard para monitorar e controlar sua placa de "Arduino ESP8266 12E" com o banco de dados "Google Firebase". Brasil: Amazon Servicos de Varejo do Brasil Ltda, 2019.

GOOGLE Play. Cleanapp. Disponível em:
<https://play.google.com/store/apps/details?id=com.kujhy.cleanapp>. Acesso em: 09 Nov. 2019.

GOOGLE Play. Lavemcasa. Disponível em:
<https://play.google.com/store/apps/details?id=com.lavemcasa.drapp>. Acesso em: 09 Nov. 2019.

GOOGLE Play. OMO Lavanderia. Disponível em:
<https://play.google.com/store/apps/details?id=br.com.taqtile.omolavanderia>.
Acesso em: 09 Nov. 2019.

GOOGLE Play. MrJeff. Disponível em: <https://play.google.com/store/apps/details?id=com.mrjefflab.app>. Acesso em: 09 Nov. 2019.

RENAN B. KALIL, Rafael A. F. Zanatta, Pedro C. B. de Paula e Beatriz Kira. Economias do Compartilhamento e o Direito. Disponivel em < http://www.internetlab.org.br/wp-content/uploads/2017/12/Economias_do_compartilhamento_e.pdf > Acesso em: 25 de out. de 2019.

PROVIDER.Disponível em:<https://pub.dev/documentation/provider/latest/provider/>. Acesso em: 01 Nov. 2020.>

Singes,Adrián Todolí. El contrato de trabajo en el S. XXI: La economía colaborativa, On-demand economy, Crowdsorcing, uber economy y otras formas de descentralización productiva que atomizan el mercado de trabajo. Disponível em: <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2705402>. Acesso em: 15 set. 2019.

SILVA, Paulo Cesar Barreto. Devmedia: 2009. Artigo SQL Magazine 64 - Utilizando UML. Disponível em:<https://www.devmedia.com.br/artigo-sql-magazine-64-utilizando-uml/12665>. Acesso em: 08 de nov. de 2019.