UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

FELIPE GUSTAVO DA CUNHA BORGES DOS SANTOS

APLICATIVO PARA ATENDER REQUISIÇÕES DE DESLOCAMENTO SOLICITADAS POR PASSAGEIROS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2020

FELIPE GUSTAVO DA CUNHA BORGES DOS SANTOS

APLICATIVO PARA ATENDER REQUISIÇÕES DE DESLOCAMENTO SOLICITADAS POR PASSAGEIROS

Projeto para conclusão do Curso de Especialização em Java da UTFPR – Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. Vinicius Pegorini

PATO BRANCO
2020

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

TERMO DE APROVAÇÃO

APLICATIVO PARA ATENDER REQUISIÇÕES DE DESLOCAMENTO SOLICITADAS POR PASSAGEIROS

por

FELIPE GUSTAVO DA CUNHA BORGES DOS SANTOS

Este trabalho de conclusão de curso foi apresentado em 09 de março de 2020, como requisito parcial para a obtenção do título de especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Vinicius Pegorini (orientador), Andreia Scariot Beulke e Beatriz Terezinha Borsoi, membros de banca. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Vinicius Pegorini
Prof. Orientador (UTFPR)

_____
Andreia Scariot Beulke
(UTFPR)

_____
Beatriz Terezinha Borsoi
(UTFPR)

_____
Vinicius Pegorini
Coordenador do curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

Com a popularização dos dispositivos móveis a população cada vez mais utiliza esses aparelhos para as mais diversas tarefas do dia a dia, tais como fazer compras e pedir alimentos, entre outras atividades. Quando uma pessoa necessita locomover-se de um ponto a outro da cidade também é possível utilizar um dispositivo móvel para solicitar esse tipo de serviço. Visando facilitar a busca por transporte, neste trabalho foi desenvolvido um aplicativo para Android que permite ao passageiro gerar uma requisição solicitando transporte a determinado local. Os motoristas disponíveis recebem a requisição e um deles pode aceitar (até que outro não aceite primeiro). Com isso, o motorista fica responsável para ir ao local do passageiro, buscá-lo e levá-lo ao local de destino.

Palavras-chave: Android. API. Firebase. Transporte.

ABSTRACT

With the popularization of mobile devices the population increasingly uses these devices for the most diverse daily tasks, such as go shopping, ordering food among others activities. When a person needs to move from one to another point in the city it is also possible use a mobile device for such a task. In order to facilitate the search for transport, an Android application was developed in this work that allows passengers to generate a transport request to a specific location. The available drivers receive the request and one of them can accept it until another does not accept it first. With that, the driver is responsible to go to the passenger's place to pick up and take him to the destination.

Keywords: Android. API. Firebase. Transport.

LISTA DE FIGURAS

Figura 1 - Diagrama de caso de uso 17
Figura 2 - Diagrama E.R do Banco de Dados 18
Figura 3 - Tela inicial do Passageiro 19
Figura 4 - Tela do Passageiro para confirmar 20
Figura 5 - Tela inicial do Motorista 21
Figura 6 - Tela do Motorista com detalhes da requisição 21
Figura 7 - Aparece para o passageiro 22
Figura 8 - A caminho do Passageiro 23
Figura 9 - Altera o mapa e mostra o destino 24
Figura 10 - Total da corrida para o Passageiro 25
Figura 11 - Motorista chegou ao destino do Passageiro 26

LISTA DE QUADROS

Quadro 1 - Ferramentas e tecnologias utilizadas 12
Quadro 2 - Requisitos Funcionais 17
Quadro 3 - Requisitos Não Funcionais 17

LISTAGEM DE CÓDIGO

Listagem 1 - Criação da tela de requisições - Método OnCreate 27
Listagem 2 - Criação da tela de requisições - Método onStart 27
Listagem 3 - Criação da tela de requisições - Método recuperarRequisicoes 28
Listagem 4 - Criação da tela de cadastros - Método validarCadastroUsuario 29
Listagem 5 - Criação da tela de cadastros - Método cadastrarUsuario 29
Listagem 6 - Criação da tela de login - Método validarLoginUsuario 31
Listagem 7 - Criação da tela de login - Método logarUsuario 31
Listagem 8 - Criação da tela de requisições para os motoristas 32

LISTA DE ABREVIATURAS E SIGLAS

BD	Banco de Dados
FRD	Firebase Realtime Database
Maps	Google Map
JDK	Java Development Kit

SUMÁRIO

1 INTRODUÇÃO 9
1.1 CONSIDERAÇÕES INICIAIS 9
1.2 OBJETIVOS 9
1.2.1 Objetivo Geral 9
1.2.2 Objetivos Específicos 10
1.3 JUSTIFICATIVA 10
1.4 ESTRUTURA DO TRABALHO 11
2 MATERIAIS E MÉTODO 12
2.1 MATERIAIS 12
2.2 Firebase Realtime Database 13
2.3 Google Maps 13
2.4 GeoFire 13
3 RESULTADOS 15
3.1 ESCOPO DO SISTEMA 15
3.2 MODELAGEM DO SISTEMA 15
3.3 APRESENTAÇÃO DO SISTEMA 18
3.3.1 Passageiro: 19
3.3.2 Motorista: 20
3.4 IMPLEMENTAÇÃO DO SISTEMA 26
4 CONSIDERAÇÕES FINAIS 34
REFERÊNCIAS 35

1 INTRODUÇÃO

Este capítulo apresenta a introdução do projeto com o escopo e o contexto do trabalho e a justificativa.

1.1 CONSIDERAÇÕES INICIAIS

Com o avanço da tecnologia dos smartphones e a popularização do acesso à internet móvel, diversos aplicativos de uso para transporte individual ou coletivo foram lançados e se tornaram parte do cotidiano da população. Aplicativos para outros segmentos também estão sendo utilizados nas mais diversas tarefas do dia a dia, tais como serviços bancários, reservas de hotéis ou até para pedir comida.

Redes de transportes são fatores essenciais na sociedade moderna, sendo responsáveis por influenciar na qualidade de vida e na eficiência de processos das cidades. Avanços no transporte proporcionam o crescimento metropolitano, cultural, nos padrões de vida dos indivíduos e são alavancados à medida que surgem novas necessidades de mobilidade e acessibilidade (GOWRISHANKAR et al, 2014).

Como a tecnologia dos dispositivos para aparelhos móveis representa uma área de grande crescimento, são disponíveis diversos modelos de aplicativos usados em diferentes modais de transporte e com uma variedade de informações inclusive do transporte urbano.

Neste cenário de crescentes inovações tecnológicas no âmbito de aplicativos de trânsito para smartphones foram desenvolvidos programas com diferentes focos, entre os quais destacam os específicos para automóveis, transporte público e bicicletas.

Nesse trabalho o foco é um aplicativo de transporte urbano específico para locomoção de usuários que solicitem uma corrida para determinado local. Nesse âmbito, ao usar o aplicativo, o usuário está fazendo uma requisição para um motorista levá-lo de um ponto "A" para um ponto "B".

1.2 OBJETIVOS

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

1.2.1 Objetivo Geral

Criar um aplicativo para dispositivos móveis para fazer a ligação entre motoristas e usuários que buscam por transporte.

1.2.2 Objetivos Específicos

- Possibilitar a busca de transporte pelo endereço de destino;
- Possibilitar calcular o tempo do trajeto;
- Apresentar ao passageiro o valor da corrida antes de iniciá-la;
- Atualizar o mapa em ambas as telas (passageiro e usuário), conforme o caminho é percorrido;
- Disponibilizar ao motorista a opção de visualizar o trajeto no Google Maps, mostrando a melhor rota e a distância atual até o destino;
- Permitir ao passageiro acompanhar o trajeto por meio do mapa principal do aplicativo;

1.3 JUSTIFICATIVA

Com o desenvolvimento tecnológico dos dispositivos móveis (smartphones, tablets), a queda nos preços e a disseminação de redes Wireless, 3G, 4G, houve crescimento elevado do mercado de aplicativos móveis (SIQUEIRA, 2012). Unindo os problemas relacionados à mobilidade e o avanço em sistemas cibernéticos foi criado um mercado de aplicativos para smartphones com componentes de posicionamento. Ao contrário dos sistemas convencionais de detecção, os aparelhos com Sistema de localização (Location Based Services - LBS) fornecem aos usuários a capacidade de associar os dados obtidos no exato momento, dividindo-os com todos os outros usuários que possuem o mesmo sistema, tornando-os ativos (FIRE et al., 2012). O LBS possibilita aos usuários reproduzir cenários atuais do transporte público por meio da relação entre a rede de dados privados de cada aparelho, com a rede pública, mapas e cadastros de rotas. Dentre as vantagens dos aplicativos baseados em geolocalização destacam-se a possibilidade de monitoramento de tráfego e sugestões de rotas, sendo viável para planejamento, gestão, operação e fiscalização do transporte urbano (CHAVES et al., 2014).

Com a popularização da Internet e devido à abundante oferta, o transporte privado feito por meio de aplicativos vem se tornando mais acessível e de grande utilização por parte das pessoas.

O foco é entregar ao passageiro comodidade para suas viagens, além de fácil solicitação das corridas pelo aplicativo, solicitando apenas o endereço de destino, o restante do fluxo é realizado automaticamente.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. O Capítulo 2 apresenta as ferramentas e as tecnologias utilizadas na modelagem e na implementação do sistema. No Capítulo 3 estão os resultados da realização do trabalho, representados por diagramas e complementações textuais, a apresentação do sistema desenvolvido por meio de suas telas e partes de código.

Por fim estão as considerações finais, definidas por conclusão, e as referências bibliográficas utilizadas na composição do texto.

2 MATERIAIS E MÉTODO

Este capítulo apresenta as ferramentas e as tecnologias utilizadas na modelagem e na implementação do aplicativo. Também é apresentada a sequência das atividades desenvolvidas para a realização do trabalho.

2.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas na modelagem e no desenvolvimento do aplicativo.

Quadro 1 - Ferramentas e tecnologias utilizadas

Ferramenta / Tecnologia	Versão	Disponível em	Aplicação
Java	JDK 1.8	https://www.java.com/pt_BR/	Linguagem para desenvolvimento da aplicação.
Android Studio	3.5.3	https://developer.android.com/	Ambiente de desenvolvimento.
Firebase Realtime Database	17.0.0	https://console.firebase.google.com/	Banco de dados.
GeoFire	2.3.1	https://github.com/firebase/geofire-java	Biblioteca de código aberto para o Firebase Realtime Database que adiciona compatibilidade com consultas geoespaciais.
Google Maps	16.1.0	https://github.com/googlemaps/	Biblioteca de geolocalização e uso de mapas.
Geocoder		https://developers.google.com/maps/documentation/geocoding	Biblioteca de conversão de endereços.

Fonte: Autoria própria.

A seguir são apresentadas, dentre as constantes no Quadro 1, as principais tecnologias utilizadas para o desenvolvimento da aplicação.

2.2 Firebase Realtime Database

O Firebase Realtime Database é um banco de dados hospedado na nuvem. Os dados são armazenados como JSON e sincronizados em tempo real com todos os clientes conectados. Quando aplicativos são criados em plataformas cruzadas com os SDKs para iOS, Android e JavaScript, todos os clientes compartilham uma instância do Realtime Database e recebem automaticamente atualizações com os dados mais recentes (FIREBASE, 2020).

2.3 Google Maps

Com o SDK do Google Maps para Android é possível adicionar mapas com base nos dados do Google Maps aos aplicativos. A API gerencia automaticamente o acesso aos servidores do Google Maps, o download de dados, a exibição do mapa e a resposta aos gestos do mapa. Também é possível usar chamadas de API para adicionar marcadores, polígonos e superposições a um mapa básico e alterar a visualização do usuário de uma área específica do mapa. Esses objetos fornecem informações adicionais para os locais do mapa e permitem a interação do usuário com o mapa (GOOGLE MAPS, 2020).

A API permite adicionar esses gráficos a um mapa:

- Ícones ancorados em posições específicas no mapa (Marcadores).
- Conjuntos de segmentos de linha (polilinhas).
- Segmentos fechados (polígonos).
- Gráficos de bitmap ancorados em posições específicas no mapa (sobreposições de solo).
- Conjuntos de imagens que são exibidas na parte superior dos blocos do mapa base (sobreposições de bloco).

2.4 GeoFire

O GeoFire é uma biblioteca de código aberto para Java que permite armazenar e consultar um conjunto de chaves com base em sua localização geográfica. Simplesmente armazena locais com teclas de sequência de caracteres. Seu principal benefício, no entanto, é a

possibilidade de consultar chaves dentro de uma determinada área geográfica, tudo em tempo real.

O GeoFire usa o banco de dados Firebase Realtime Database para armazenamento de dados, permitindo que os resultados da consulta sejam atualizados em tempo real à medida que mudam. Carrega seletivamente apenas os dados perto de determinados locais, mantendo seus aplicativos leves e responsivos, mesmo com conjuntos de dados extremamente grandes (GEOCODING API, 2020).

3 RESULTADOS

Este capítulo apresenta o resultado da realização do trabalho, um aplicativo desenvolvido para permitir gerar solicitações de transporte para um determinado local.

3.1 ESCOPO DO SISTEMA

O aplicativo desenvolvido neste trabalho é dividido em dois tipos de cadastros com suas funcionalidades específicas:

a) Passageiro - É quem faz a solicitação da corrida e apenas visualiza no mapa o local onde o motorista está, com atualização em tempo real de sua localização. O passageiro somente cria a requisição de corrida, porém, não tem acesso aos dados, somente ao mapa.

Após o passageiro realizar o cadastro no aplicativo, ele poderá solicitar uma corrida, criando uma requisição. Ao realizar a requisição, o aplicativo grava a localização atual onde o passageiro se encontra e possibilita por meio de uma caixa de texto, que seja informada a busca pelo local de destino (que o passageiro deseja ir). A busca é feita por (cidade, CEP, bairro, rua e número).

b) Motorista - Têm como principal a tela de requisições que apresenta todas as solicitações de corridas dos passageiros. As informações disponíveis para o motorista nessa tela são o nome do passageiro e a distância atual. O motorista pode clicar na requisição e se tiver mais de uma requisição, o motorista pode escolher a corrida que deseja aceitar (basta que aceite primeiro que outros motoristas), mas não é obrigado a aceitar nenhuma das requisições que aparecer na tela, ele pode recusar, porém, se aceitar não há como cancelar. Quando aceita a requisição, é aberto o mapa com foco na localização do passageiro e já carrega na tela um círculo em laranja, esse círculo têm como objetivo facilitar para o motorista encontrar o passageiro, pois, o círculo significa um marcação de 50 metros, ou seja, o passageiro está dentro da zona do círculo. Quando o motorista chega até o raio laranja (50 metros do passageiro) significa que está muito próximo do passageiro, então, automaticamente desaparece o círculo em laranja e aparece o destino no mapa, ou seja, o local para o qual o passageiro solicitou a corrida.

3.2 MODELAGEM DO SISTEMA

Os usuários que possuem acesso ao sistema pertencem a duas categorias: passageiros ou motoristas.

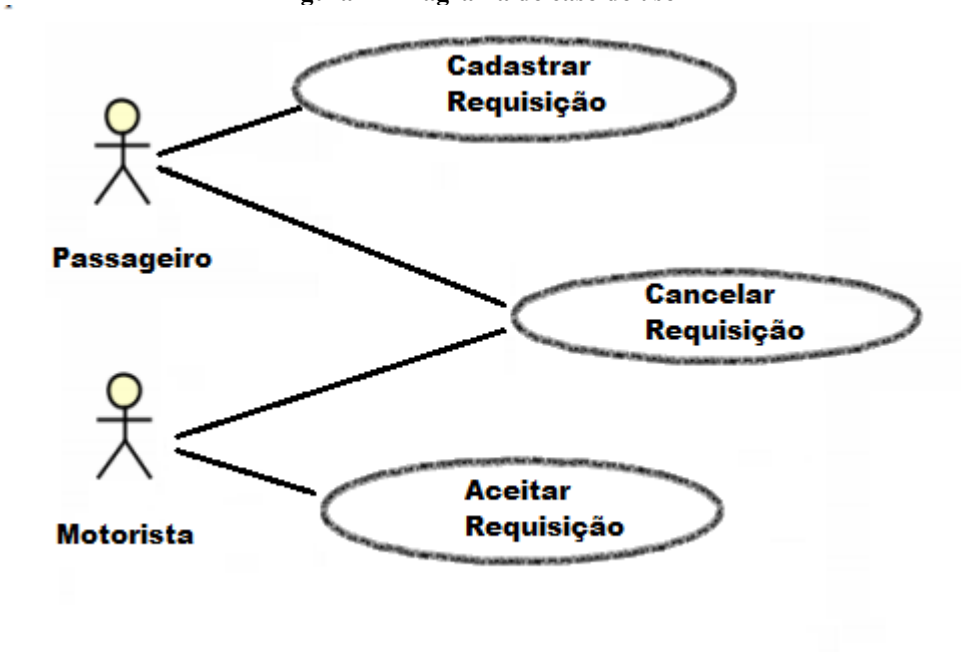
Passageiro é quem cria as corridas. Motorista é o responsável por realizar a corrida, ou seja, levar o passageiro onde ele solicitar.

O sistema possui as seguintes telas:

- Tela de cadastro (Passageiro/Motorista) - ao efetuar o cadastro há a opção para definir se o usuário será um Passageiro ou Motorista.
- Tela de login: local em que são preenchidos os dados cadastrados, para conseguir acessar o aplicativo.
- Tela do passageiro: diferente da tela do motorista, pois, a tela do passageiro possui um campo de texto no qual é preenchido o local da corrida.
- Tela de requisições: tela principal do motorista, nela aparecem as solicitações de corridas dos passageiros, com a distância e a descrição do local.
- Tela de corrida: é um mapa que no primeiro momento aparece o motorista por meio de um ícone de carro e o passageiro, por um ícone de um avatar de pessoa. No raio da pessoa possui uma zona em laranja, quando o motorista entra nessa zona, muda o mapa e aparece o local de destino da corrida, ou seja, local que o passageiro solicitou a corrida.

Na Figura 1 é apresentado o Diagrama de caso de uso, utilizado para descrever as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema. Casos de uso são tipicamente relacionados a "atores". Um ator é uma entidade que interage com o sistema para executar um procedimento. Nesse diagrama há dois atores: Passageiro e Motorista.

Figura 1 - Diagrama de caso de uso



Fonte: Autoria própria.

O Quadro 2 apresenta os requisitos funcionais. Os requisitos funcionais definem o que o sistema deve fazer ou o que deve atender.

Quadro 2 - Requisitos Funcionais

Identificação	Nome	Descrição
01	Cadastrar requisição	O passageiro pode cadastrar requisições
02	Aceitar requisição	O motorista pode aceitar requisições
03	Cancelar requisição	O passageiro pode cancelar a requisição que ele gerou no momento em que ela foi criada e antes que o motorista aceite a requisição. Já o motorista pode abrir a requisição gerada pelo passageiro, e se não quiser aceitar, ele pode cancelar.

Fonte: Autoria própria.

O Quadro 3 apresenta os requisitos não funcionais. Os requisitos não funcionais são praticamente todas as necessidades que não podem ser atendidas através de funcionalidades. Definem características como método de desenvolvimento, tempo e ou espaço.

Quadro 3 - Requisitos Não Funcionais

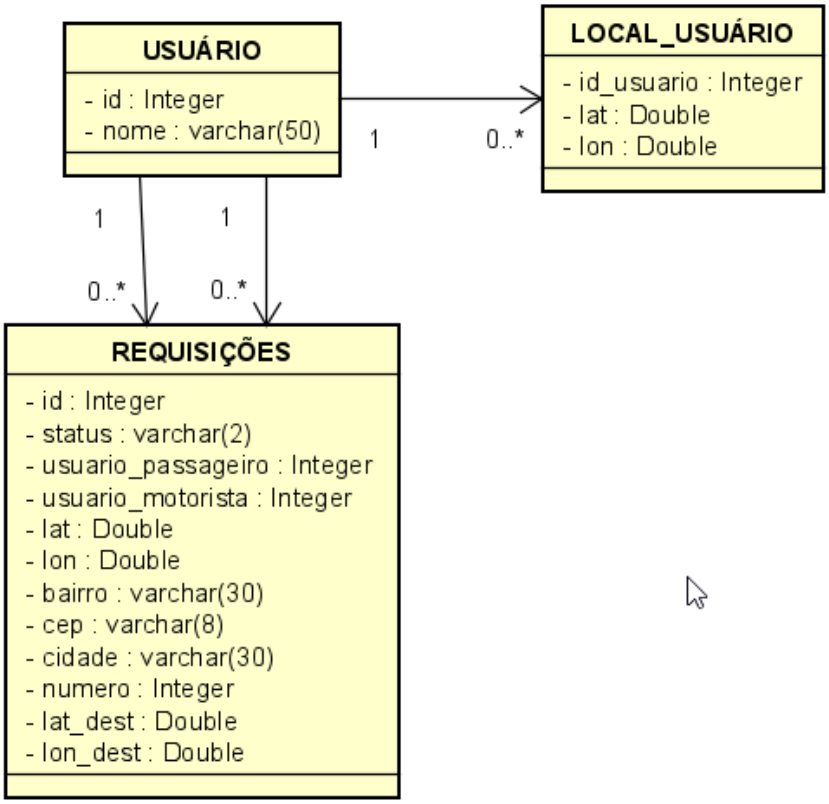
Identificação	Nome	Descrição
01	Lista de requisições	O passageiro não tem acesso à lista de requisições, ele visualiza somente a requisição que criou.

Fonte: Autoria própria.

O diagrama de entidade-relacionamento apresentado na Figura 2 representa a maneira como o banco de dados da aplicação foi criado. Possui três tabelas que se relacionam e elencam

as entidades, os campos principais, tipos de dados e os relacionamentos existentes entre as tabelas.

Figura 2 - Diagrama E.R do Banco de Dados



Fonte: Autoria própria.

3.3 APRESENTAÇÃO DO SISTEMA

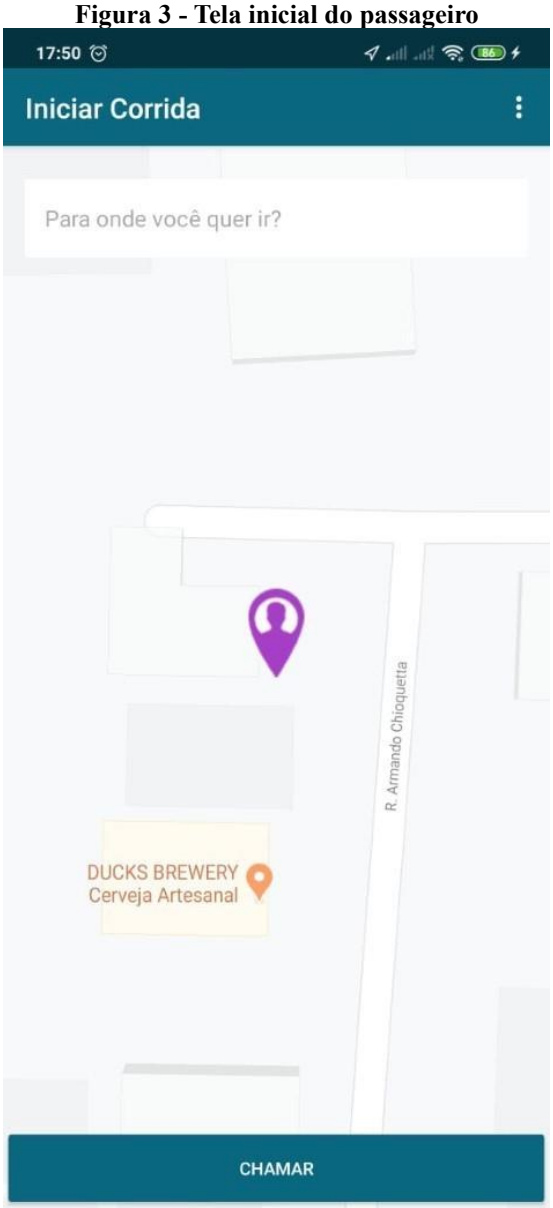
A tela inicial após o login é diferente para dois tipos de usuário, sendo composta por dois leiautes. A diferença está na validação do tipo de cadastro (passageiro ou motorista). Conforme o cadastro do usuário, é aberta a activity específica para suas finalidades.

A finalidade do passageiro é solicitar uma corrida, enquanto, a finalidade do motorista é aceitar uma corrida.

3.3.1 Passageiro

A Figura 3 apresenta a tela inicial do sistema para o passageiro. O conteúdo é basicamente a tela para preencher o local aonde o passageiro deseja ir e um botão para entrar em contato com o motorista.

Figura 3 - Tela inicial do passageiro



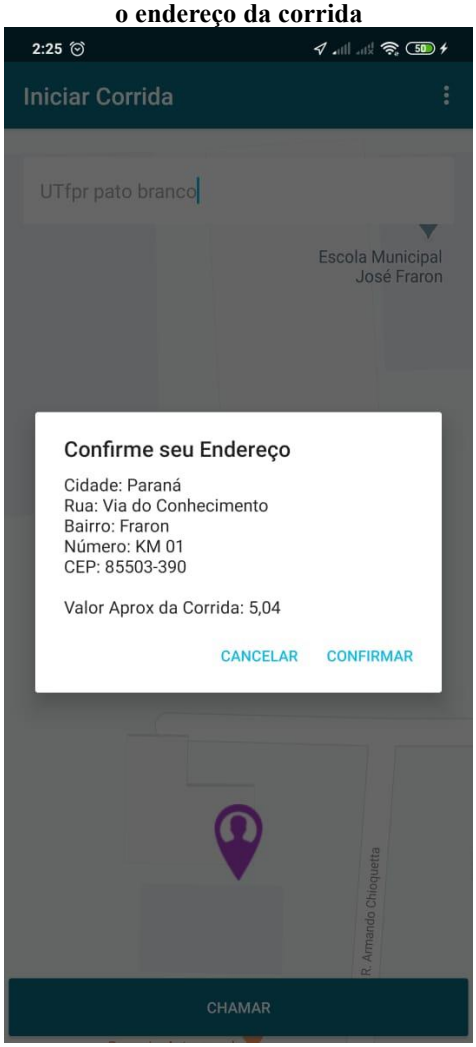
Fonte: Autoria própria.

A Figura 4 apresenta a tela de confirmação de endereço para criar a requisição de corrida por parte do passageiro. Nessa tela são apresentadas as informações de endereço do

passageiro, como: Cidade, Rua, Bairro, Número e CEP. Dados que são úteis ao motorista, para levar o passageiro na localização exata que ele deseja.

Também nesta tela consta o valor aproximado da corrida, considerando a localização atual do passageiro e a localização informada para destino.

Figura 4 - Tela do passageiro para confirmar o endereço da corrida

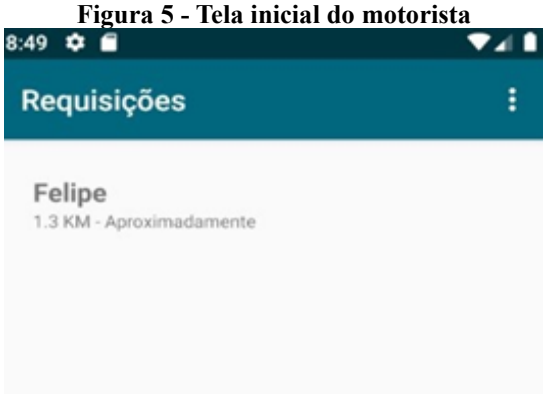


Fonte: Autoria própria.

3.3.2 Motorista

A Figura 5 apresenta a tela inicial do sistema para o motorista. Essa tela apresenta a lista de requisições pendentes com informações da distância e o nome do passageiro.

Figura 5 - Tela inicial do motorista



Fonte: Autoria própria.

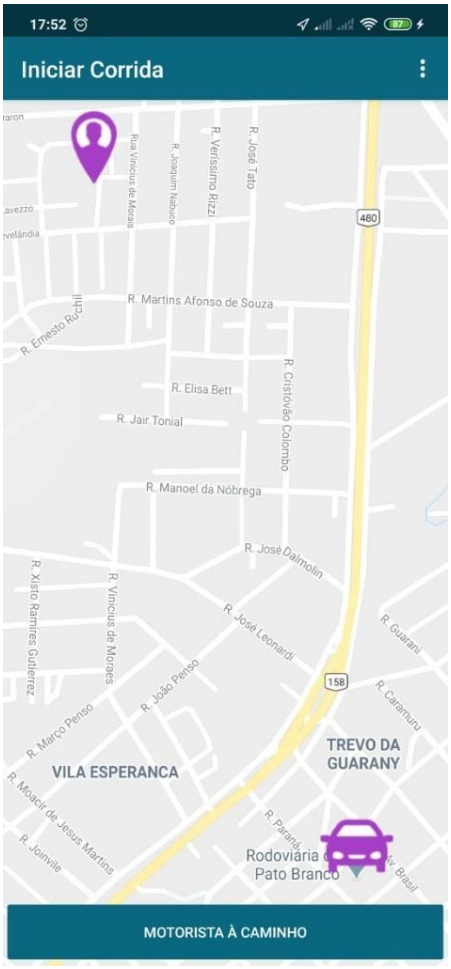
A Figura 6 apresenta a tela do motorista com os detalhes da corrida, mostra no mapa o local onde o passageiro solicitou para ser transportado e possui o botão “ACEITAR”, para que o motorista fique responsável pela requisição e tenha a obrigação de buscar o passageiro e levá-lo ao seu destino.

Figura 6 - Tela do motorista com detalhes da requisição e a opção de aceitar a corrida

Fonte: Autoria própria.

A Figura 7 mostra no mapa a localização do motorista que aceitou a corrida no ponto em que se encontra e mostra também o ponto que o passageiro está localizado. Nessa etapa, conforme o motorista se move, é atualizado no mapa em tempo real.

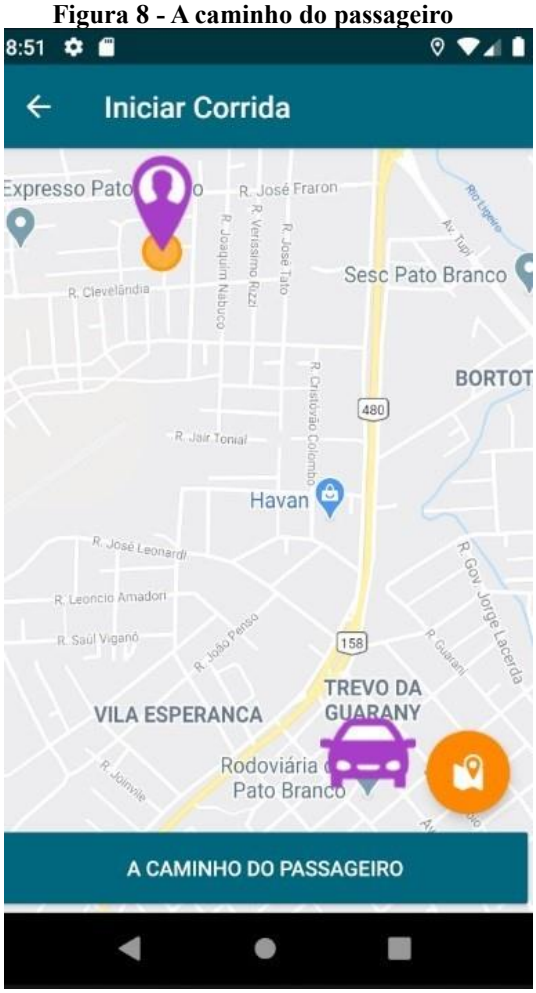
Figura 7 - Aparece para o passageiro o motorista a caminho



Fonte: Autoria própria.

A tela apresentada na Figura 8 é exibida para o motorista, aparece em laranja o raio de 50 metros da distância aproximada do passageiro e mostra no canto inferior da tela, em laranja com branco uma opção de mapa, essa com funcionalidade de abrir o aplicativo Google Maps do celular e calcular a distância do motorista e a melhor rota.

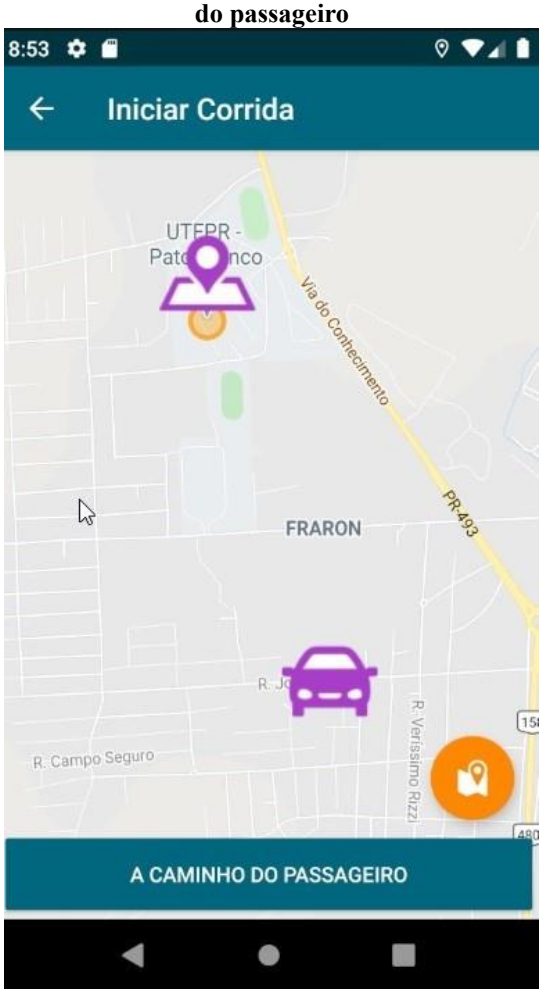
Figura 8 - A caminho do passageiro



Fonte: Autoria própria.

A Figura 9 mostra o ponto de destino do passageiro, com o ícone do Maps para calcular a melhor rota e mostrar a distância. Para alterar a tela anterior para essa, é necessário que o motorista chegue ao encontro do passageiro (raio de 50 m).

Figura 9 - Altera o mapa e mostra o destino do passageiro



Fonte: Autoria própria.

A Figura 10 mostra o valor total da corrida para o passageiro destacada na tela e, na parte inferior da tela, exibe o valor e a distância percorrida.

Figura 10 - Total da corrida para o passageiro

Fonte: Autoria própria.

A Figura 11 apresenta uma corrida finalizada, o valor e a distância percorrida. Essa tela é apresentada ao motorista quando ele chega ao destino do passageiro.

Figura 11 - Motorista chegou ao destino do passageiro

Fonte: Autoria própria.

3.4 IMPLEMENTAÇÃO DO SISTEMA

Na Listagem 1, ao criar a activity são instanciados dois métodos: um para inicializar os componentes chamado inicializarComponentes(), nele são instanciados os construtores e o outro método possui um serviço para recuperar a localização do usuário chamado recuperarLocalizacaoUsuario(), recuperando do banco de dados a latitude e a longitude do passageiro.

Listagem 1 - Criação da tela de requisições - Método OnCreate

```
protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_requisicoes);

        inicializarComponentes();

        recuperarLocalizacaoUsuario();
```

Fonte: Autoria própria.

A Listagem 2 possui o método chamado de verificarStatusRequisicao(),utilizado primeiramente para recuperar do banco de dados o usuário autenticado. Então, o método acessa o nó de requisições, valida se a requisição está pendente e abre uma nova activity que é a tela da corrida, por meio do método abrirTelaCorrida().

Listagem 2 - Criação da tela de requisições - Método onStart

```
  protected void onStart() {
      super.onStart();
      verificaStatusRequisicao();
  }

  private void verificaStatusRequisicao(){
      Usuario usuarioLogado = UsuarioFirebase.getDadosUsuarioLogado();
      DatabaseReference firebaseRef =
      ConfiguracaoFirebase.getFirebaseDatabase();
      DatabaseReference requisicoes = firebaseRef.child("requisicoes");
      Query requisicoesPesquisa = requisicoes.orderByChild("motorista/id")
          .equalTo(usuarioLogado.getId());
      requisicoesPesquisa.addListenerForSingleValueEvent(new
  ValueEventListener() {

    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
       for(DataSnapshot ds: dataSnapshot.getChildren()) {
          Requisicao requisicao = ds.getValue(Requisicao.class);
          if (requisicao.getStatus().equals(Requisicao.STATUS_A_CAMINHO)
           || requisicao.getStatus().equals(Requisicao.STATUS_VIAGEM)
           || requisicao.getStatus().equals(Requisicao.STATUS_FINALIZADA)){
              motorista = requisicao.getMotorista();
             abrirTelaCorrida(requisicao.getId(), motorista, true);
       }
     }
}
```

Fonte: Autoria própria.

Na Listagem 3, o método recuperarRequisicoes() valida se a situação estiver como AGUARDANDO e por meio do DataSnapshot recebe o valor da requisição e depois adiciona

em um conjunto de requisições que no caso é apresentado em um recyclerView, que é um componente utilizado para carregar informações em lista, uma abaixo da outra.

Listagem 3 - Criação da tela de requisições - Método recuperarRequisicoes

```
private void recuperarRequisicoes(){

    DatabaseReference requisicoes = firebaseRef.child("requisicoes");

    Query requisicaoPesquisa = requisicoes.orderByChild("status")
            .equalTo(Requisicao.STATUS_AGUARDANDO);

    requisicaoPesquisa.addValueEventListener(new ValueEventListener() {

        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (dataSnapshot.getChildrenCount() > 0){
                textResultado.setVisibility(View.GONE);
                recyclerRequisicoes.setVisibility(View.VISIBLE);
            }else {
                textResultado.setVisibility(View.VISIBLE);
                recyclerRequisicoes.setVisibility(View.GONE);
            }
            listaRequisicoes.clear();
            for(DataSnapshot ds: dataSnapshot.getChildren()){
                Requisicao requisicao = ds.getValue(Requisicao.class);
                listaRequisicoes.add(requisicao);
            }
            adapter.notifyDataSetChanged();
        }
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
        }
    });
}
```

Fonte: Autoria própria.

Na Listagem 4 está o método validarCadastroUsuario() que têm por finalidade receber os dados informados nos campos da tela, validar se foram preenchidos corretamente e chamar o método cadastrarUsuario() para autenticar os dados e salvá-los no banco de dados.

Listagem 4 - Criação da tela de cadastros - Método validarCadastroUsuario

```
public void validarCadastroUsuario(View view) {
    String textoNome = campoNome.getText().toString();
    String textoEmail = campoEmail.getText().toString();
    String textoSenha = campoSenha.getText().toString();

    if (!textoNome.isEmpty()) {
        if (!textoEmail.isEmpty()) {
            if (!textoSenha.isEmpty()) {
                Usuario usuario = new Usuario();
                usuario.setNome(textoNome);
                usuario.setEmail(textoEmail);
                usuario.setSenha(textoSenha);
                usuario.setTipo(verificaTipoUsuario());

                cadastrarUsuario(usuario);
            }else{
                Toast.makeText(CadastroActivity.this,
                       "Preencha a Senha",
                        Toast.LENGTH_SHORT).show();
            }
        }else{
                 Toast.makeText(CadastroActivity.this,
                        "Preencha o E-mail",
                         Toast.LENGTH_SHORT).show();
        }
    }else{
                Toast.makeText(CadastroActivity.this,
                        "Preencha o Nome",
                        Toast.LENGTH_SHORT).show();
  }
}
```

Fonte: Autoria própria.

Na Listagem 5, o método cadastrarUsuario() faz a autenticação com o banco de dados e chama o método createUserWithEmailAndPassword(), para obrigar informar o e-mail e senha de acordo com o padrão do Firebase. Após isso ocorre a validação, em caso de sucesso os dados são armazenados. Por fim, verificar qual é o tipo de usuário e mostra na tela um Toast com a informação “Tipo de usuário Cadastrado”.

Listagem 5 - Criação da tela de cadastros - Método cadastrarUsuario

```
public void cadastrarUsuario(final Usuario usuario){
    autenticacao = ConfiguracaoFirebase.getFirebaseAutenticacao();
    autenticacao.createUserWithEmailAndPassword(
            usuario.getEmail(),
            usuario.getSenha()
       ).addOnCompleteListener(this,new OnCompleteListener<AuthResult>() {
```

```
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
     if(task.isSuccessful()){
       try{
         String idUsuario =  task.getResult().getUser().getUid();
        usuario.setId(idUsuario);
       usuario.salvar();

       UsuarioFirebase.atualizarNomeUsuario(usuario.getNome());
       if(verificaTipoUsuario() == "P"){
          startActivity(new Intent(CadastroActivity.this,
          PassageiroActivity.class));
          finish();
          Toast.makeText(CadastroActivity.this,
                        "Passageiro Cadastrado!",
                        Toast.LENGTH_SHORT).show();
       }else{
          startActivity(new Intent(CadastroActivity.this,
          RequisicoesActivity.class));
          finish();
          Toast.makeText(CadastroActivity.this,
                       "Motorista Cadastrado!",
                        Toast.LENGTH_SHORT).show();
       }

      }catch (Exception e){
        e.printStackTrace();
      }
    }else {
       String excecao = "";
       try {
          throw task.getException();
       }catch (FirebaseAuthWeakPasswordException e){
           excecao = "Senha Fraca!";
       }catch (FirebaseAuthInvalidCredentialsException e){
           excecao = "E-mail Inválido!";
       }catch (FirebaseAuthUserCollisionException e){
           excecao = "Conta Cadastrada!";
       }catch (Exception e){
           excecao = "Falha ao Cadastrar Usuário " +
           e.getMessage();
           e.printStackTrace();
       }
       Toast.makeText(CadastroActivity.this,
                      excecao,
                      Toast.LENGTH_SHORT).show();
    }
  }});
}
```

Fonte: Autoria própria.

Na Listagem 6 é verificado se nenhum campo da tela está em branco, após isso, é chamado o método logarUsuario().

Listagem 6 - Criação da tela de login - Método validarLoginUsuario

```
public void validarLoginUsuario(View view){
    String textoEmail = campoEmail.getText().toString();
    String textoSenha = campoSenha.getText().toString();
    if(!textoEmail.isEmpty()){
        if(!textoSenha.isEmpty()){
            Usuario usuario = new Usuario();
            usuario.setEmail(textoEmail);
            usuario.setSenha(textoSenha);
            logarUsuario(usuario);
        }else{
            Toast.makeText(LoginActivity.this,
                    "Preencha Senha",
                    Toast.LENGTH_SHORT).show();
        }
    }else{
        Toast.makeText(LoginActivity.this,
                "Preencha E-mail",
                Toast.LENGTH_SHORT).show();
  }
 }
}
```

Fonte: Autoria própria.

Na Listagem 7 é apresentado o método logarUsuario(). Esse método é utilizado para realizar a autenticação dos dados por meio do método signInWithEmailAndPassword após isso, acessa o método redirecionaUsuarioLogado(). Esse método valida os campos no banco de dados para comparar os dados. Em seguida, conforme o tipo de usuário é aberta a Activity. Ou seja, caso seja do tipo "M " é aberta a RequisicoesActivity, e caso seja do tipo "P " é aberta a PassageiroActivity.

Listagem 7 - Criação da tela de login - Método logarUsuario

```
public void logarUsuario(Usuario usuario){
    autenticacao = ConfiguracaoFirebase.getFirebaseAutenticacao();
    autenticacao.signInWithEmailAndPassword(
            usuario.getEmail(),usuario.getSenha()
    ).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
  if (task.isSuccessful()){
     UsuarioFirebase.redirecionaUsuarioLogado(LoginActivity.this);
  }else{
     String excecao = "";
          try                                                    {
             throw task.getException();
          }catch (FirebaseAuthInvalidUserException e){
             excecao = "Usuário não Cadastrado!";
          }catch (FirebaseAuthInvalidCredentialsException e){
             excecao = "E-mail ou Senha Inválido!";
          }catch (Exception e){
```

```
                excecao = "Falha ao Acessar" + e.getMessage();
                e.printStackTrace();
            }
            Toast.makeText(LoginActivity.this,
                           excecao,
                           Toast.LENGTH_SHORT).show();
            }
    }
   });
}
```

Fonte: Autoria própria.

Na Listagem 8, se a corrida não estiver com status de cancelada, então, é instanciada a classe addressDestino. Essa classe recebe o método recuperarEndereco que utiliza um Geocoder para transformar o endereço digitado em coordenadas geográficas. Com as coordenadas geográficas são utilizados os métodos da classe addressDestino para atribuir as respectivas informações. Após isso é mostrado na tela um Alert com os dados da corrida, que o passageiro solicitou. Caso confirme, é gerada a requisição para o motorista.

Listagem 8 - Criação da tela de requisições para os motoristas

```
public void chamarRunner(View view){
    //false -> Corrida Não Pode ser Cancelada
    //true -> Corrida Pode ser Cancelada
    if(cancelarCorrida){
        //Cancelar a Requisição
        requisicao.setStatus(Requisicao.STATUS_CANCELADA);
        requisicao.atualizarStatus();
    }else {
       String enderecoDestino = editDestino.getText().toString();
       if(!enderecoDestino.equals("") || enderecoDestino != null){
           Address addressDestino = recuperarEndereco(enderecoDestino);
           if (addressDestino != null){

            final Destino destino = new Destino();
            destino.setCidade(addressDestino.getAdminArea());
            destino.setCep(addressDestino.getPostalCode());
            destino.setBairro(addressDestino.getSubLocality());
            destino.setRua(addressDestino.getThoroughfare());
            destino.setNumero(addressDestino.getFeatureName());
       destino.setLatitude(String.valueOf(addressDestino.getLatitude()));
            destino.setLongitude(String.valueOf(addressDestino.getLongit
ude()));
            StringBuilder mensagem = new StringBuilder();
            mensagem.append("Cidade: " + destino.getCidade());
            mensagem.append("\nRua: " + destino.getRua());
            mensagem.append("\nBairro: " + destino.getBairro());
            mensagem.append("\nNúmero: " + destino.getNumero());
            mensagem.append("\nCEP: " + destino.getCep());
AlertDialog.Builder builder = new AlertDialog.Builder(this)
      .setTitle("Confirme seu Endereço")
      .setMessage(mensagem)
```

```
      .setPositiveButton("Confirmar", new
DialogInterface.OnClickListener() {

 @Override
 public void onClick(DialogInterface dialog, int which) {
  salvarRequisicao(destino);
}
}).setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {

  @Override
  public void onClick(DialogInterface dialog, int which) {
   }
   });
   AlertDialog dialog = builder.create();
  dialog.show();

  }
        }else{
            Toast.makeText(this,
                    "Informe o Endereço",
                    Toast.LENGTH_SHORT).show();
   }

  }

}
```

Fonte: Autoria própria.

4 CONSIDERAÇÕES FINAIS

O desenvolvimento do aplicativo, das funcionalidades propostas e documentadas para o controle de requisições de corridas, foi realizado conforme planejado, atendendo o foco e os objetivos propostos. Foram utilizadas tecnologias para as funcionalidades do aplicativo, dentre elas, a linguagem Java e Android, que permitem o uso de várias bibliotecas úteis para implementação.

Os objetivos propostos foram realizados e o aplicativo foi desenvolvido. Ele permite o controle de requisições das corridas por parte dos motoristas, da solicitação pelo passageiro até a finalização da corrida.

O acesso é simples por meio de e-mail e senha cadastrados, de acordo com o padrão. A diferença no cadastro é na parte de definir se será Motorista ou Passageiro, pois, de acordo com isso, haverá mudança nas telas, ao acessar o aplicativo.

As tecnologias utilizadas foram as principais do Google, conforme pesquisas, foram as mais recomendadas, por fácil utilização no aplicativo e variado conteúdo na internet. Com isso, facilitou o desenvolvimento, há muito material e exemplos prontos para realizar a codificação. Até trechos de códigos com os exemplos, a base de dados é muito completa. As maiores dificuldades encontradas foram para converter endereços em coordenadas geográficas, porém, através de pesquisas, fora encontrada a biblioteca própria para realizar a conversão e com isso facilitou muito em todo o desenvolvimento. A maior parte foi lógica e tratamentos específicos para cada tela, sendo que, outra dificuldade foram os mapas, mas pesquisando, encontra-se tudo o que é necessário para o desenvolvimento.

Os trabalhos futuros para melhoria no aplicativo são o desenvolvimento de uma funcionalidade para armazenar as corridas concluídas, pois, no momento ao encerrar a corrida, não aparece o histórico na tela, tanto para o motorista como também para o passageiro. A segunda funcionalidade que será implementada é uma forma para analisar o cadastro do motorista e do passageiro, validando CNH do motorista, RG do passageiro e outras informações necessárias que serão avaliadas, para aceitá-los ou não no aplicativo.

REFERÊNCIAS

JAVA. Linguagem de Programação. Disponível em: <https://www.java.com/pt_BR/>. Acesso em: 06 jan 2020.

GEOCODING API. Biblioteca de conversão. Disponível em: < https://developers.google.com/maps/documentation/geocoding>. Acesso em: 09 jan 2020.

GOOGLE MAPS. Biblioteca de Mapas. Disponível em: <https://cloud.google.com/maps-platform https://developers.google.com/maps/documentation>. Acesso em: 10 jan 2020.

FIREBASE. Banco de dados. Disponível em: <https://firebase.google.com/>. Acesso em: 07 jan 2020.