UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

CLEVERSON MACHADO

APLICATIVO ANDROID PARA MAPEAMENTO DE ROTAS PERCORRIDAS E SITE PARA GERENCIAMENTO E VISUALIZAÇÃO DAS ROTAS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2019

CLEVERSON MACHADO

APLICATIVO ANDROID PARA MAPEAMENTO DE ROTAS PERCORRIDAS E SITE PARA GERENCIAMENTO E VISUALIZAÇÃO DAS ROTAS

Monografia de especialização apresentada na disciplina de Metodologia da Pesquisa, do Curso de Especialização em Tecnologia Java, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2019

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

UTFPR

TERMO DE APROVAÇÃO

APLICATIVO ANDROID PARA MAPEAMENTO DE ROTAS PERCORRIDAS E SITE PARA GERENCIAMENTO E VISUALIZAÇÃO DAS ROTAS

por

CLEVERSON MACHADO

Este trabalho de conclusão de curso foi apresentado em 23 de novembro de 2019, 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 Beatriz Terezinha Borsoi (orientadora), Andreia Scariot Beulke e Vinicius Pegorini, membros de banca. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Beatriz Terezinha Borsoi
Prof. Orientador (UTFPR)

_____
Andreia Scariot Beulke
(UTFPR)

_____
Vinicius Pegorini
(UTFPR)

_____
Vinicius Pegorini
Coordenador do curso

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

AGRADECIMENTOS

A Deus por mais essa conquista alcançada.

Aos meus colegas e amigos junto com os quais tive o prazer de compartilhar sábados de intensas aulas.

À minha Orientadora, professora Beatriz T. Borsoi, pela confiança, paciência e maestria.

A todos os excelentíssimos professores, com quem tive a oportunidade de conviver e aprender, que não mediram esforços para transmitirem seus conhecimentos de forma fácil e didática.

A minha família, e a todos aqui não citados, mas que de alguma forma incentivaram e colaboraram com essa conquista.

RESUMO

Os dispositivos móveis possuem uso amplo e diversificado e dados estatísticos confirmam a existência de elevado número desses dispositivos. Esses equipamentos foram originalmente criados para realizar ligações telefônicas, mas atualmente eles estão sendo utilizados para as mais diversas finalidades: de entretenimento e como ferramenta de trabalho. Recursos oferecidos por esses equipamentos, que foram planejados, inicialmente, para telefonia móvel, atualmente incluem: câmeras, acelerômetro, *Global Positioning System* (GPS), leitura de impressão digital, sensores e dispositivos de reconhecimento de movimentos, entre outros. Considerando a grande quantidade desses dispositivos e os muitos recursos que eles oferecem, neste trabalho foi desenvolvido de um aplicativo Android para mapeamento de rotas percorridas pelo usuário e um aplicativo *web* para gerenciamento e compartilhamento de rotas percorridas entre os usuários do sistema. O aplicativo pode ser utilizado, por exemplo, por esportistas e amadores que percorrem caminhos (como trilhas) desconhecidos, auxiliando-os a retornar dessas trilhas com segurança, refazer o trajeto em outras ocasiões ou divulgar esses trajetos para que outras pessoas possam fazê-los com segurança e mais tranquilidade. O aplicativo permite a inclusão de fotos e marcação de pontos e a aplicação *web* de comentários. Vinculado ao aplicativo *mobile* está uma aplicação *web* para armazenamento dessas rotas, de maneira que outros usuários possam baixá-las em seus aplicativos e percorrê-las. Esse uso não se aplica somente a lugares de difícil localização, como pântanos, matas e regiões desérticas, mas pode, também, ser usado por turistas em cidades e localidades desconhecidas. Além das finalidades citadas, o aplicativo pode ser utilizado para registrar caminhos percorridos, como roteamento de frotas e de entregas. O aplicativo desenvolvido permite definir a distância entre as marcações dos pontos da rota (a cada quantos metros um marcador será inserido) ou em tempo (a cada quantos minutos inserir um marcador da rota). Além dos marcadores, é possível armazenar fotos durante o percurso, auxiliando, assim, na identificação do local com o objetivo que a rota possa ser refeita pela pessoa ou por outras, para divulgação se a rota tiver finalidade turística, para comprovação se for vinculada a trabalho ou outros.

Palavras-chave: Aplicação Android. Dispositivos móveis. Aplicações *web*.

ABSTRACT

Mobile devices have wide and diverse use and statistical data confirm the high quantity of these devices. These devices were created to make telephone calls, but today they are used for a variety of purposes: from entertainment to work tool. Nowadays features offered by these devices, initially designed for mobile telephony, include, but are not limited to, cameras, accelerometer, Global Positioning System (GPS), fingerprint reading, sensors and motion recognition devices. Considering the existence of a large number of these devices and the many features they offer, it was develop an Android application for mapping user-traveled routes. The application can be used, for example, by sportsmen and amateurs who travel in unknown paths (such as trails) and safely return from these trails or retrace them or to publicize the route. Linked to the mobile application developed is a web application for storing these routes so that other users can download them and use them as a guide. This use not only applies to help trekking in places such as swamps, forests and desert regions, but also can be used by tourists in unfamiliar cities and locations. In addition to the aforementioned purposes, the application can be used to record paths traveled, such as fleet routing and delivery. The application developed allows marking the points of the route by distance (every how many meters a marker will be inserted) or time (every how many minutes insert a marker of the route). In addition, it is possible to store photos during the route. Thus, assisting in the identification of the place when the purpose is the route could be redone, for disclosure if the route has a tourist purpose, for proof if it is linked to work or others.

Keywords: Android application. GPS. Mobile devices. Web Apps.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

API	*Application Programming Interface*
CDMA	*Code Division Multiple Access*
CRM	*Customer Relationship Management*
DAO	*Data Access Object*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
JSON	*JavaScript Object Notation*
REST	*Representational state transfer*
RF	Requisito Funcional
RNF	Requisito Não Funcional
TI	Tecnologia de Informação
URL	*Uniform Resource Locator*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

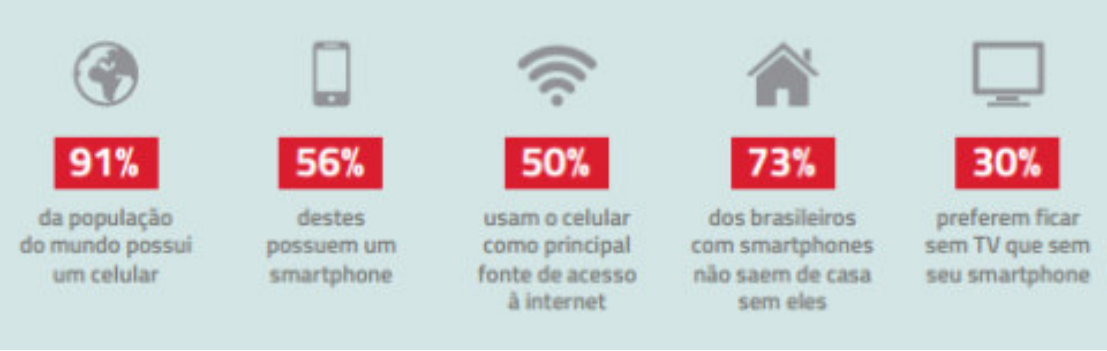
Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa do trabalho. O capítulo é finalizado com uma breve apresentação dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

Em 2013, o número de celulares no mundo alcançou 6,8 bilhões, representando 96% do total da população (GLOBAL..., 2014), se considerado apenas um aparelho por habitante. Contudo, é comum que pessoas possuam *chip* de operadoras distintas em um mesmo aparelho celular e que uma pessoa possua mais de um aparelho. Em 2016, 1,5 bilhão de *smartphones* foram vendidos no mundo (MOBILE TIME, 2016). Dados da Anatel indicaram que no Brasil em agosto de 2019 havia 228,2 milhões de telefones celulares, com densidade de 108,28 celulares por 100 habitantes (ANATEL, 2019).

A Figura 1 apresenta percentuais de utilização de *smartphones* pela população mundial. Esses dados indicam que 91% da população mundial possui telefone celular e mais da metade desses telefones são do tipo *smartphone*.

Figura 1 – Uso de dispositivos móveis



Fonte: Cnova Marktplace (2015, p.s.n.).

Entre os dispositivos móveis (*smartphones* e *tablets*) mais populares estão (PAVLIĆ; PAVLIĆ; JOVANOVIĆ, 2012): iPhone, iPad, Eee Pad, Blueberry e Slate PC. Esses mesmos autores destacam que no campo dos *smartphones* quatro sistemas operacionais dominam atualmente o mercado: Symbian (Nokia), Android (Google), iOS (Apple) e Windows Mobile (Microsoft).

As tecnologias móveis tais como os *smartphones* estão definindo uma nova geração de consumidores e de aplicações para negócio (TENG; HELPS, 2010). Esses autores destacam que a partir do surgimento da *web*, propiciando o uso de *email* para acesso remoto a *Customer Relationship Management* (CRM), não há limite para o que pode ser realizado com os dispositivos móveis.

Para os profissionais de Tecnologia de Informação (TI) esse nível sem precedentes de mobilidade é tanto uma oportunidade como um desafio (TENG; HELPS, 2010). A adoção de tecnologias móveis pode ser um processo complexo, não somente pela diversidade de dispositivos existentes com um crescente número de sistemas operacionais (como, por exemplo, BlackBerry OS, Windows Mobile, Symbian OS e iPhone OS), mas, também, porque desenvolver aplicativos para esses dispositivos envolve diversos sistemas *backend* e uma variedade de tecnologias, como, *Code Division Multiple Access* (CDMA), *Global System for Mobile communications* (GSM), Wi-Fi, WiMax, 3G e 4G *networks*.

Os dispositivos móveis também podem ser utilizados para finalidades de geolocalização. Aplicativos como Google Maps fazem uso de *Global Positioning System* (GPS) e os aplicativos que executam nos dispositivos móveis podem utilizar recursos que permitem obter a localização geográfica atual do usuário, por exemplo. Duas formas básicas de localização são consideradas atualmente: localização por GPS e triangulação do sinal GSM (EICHLER; LUKE; REUFENHEUSER, 2009).

O GPS é um sistema de posicionamento que tem como base os sinais enviados por uma rede de satélites. O receptor GPS calcula o tempo que o sinal, de pelo menos três satélites, leva para chegar e, a partir desse tempo calcula sua posição com probabilidade de erro de poucos metros (MORIMOTO, 2010). A localização por triangulação de sinal GSM consiste em fornecer a localização atual do usuário tendo como base a intensidade do sinal deste para com as torres próximas. Quanto mais torres existirem próximas à localização do usuário, maior será a precisão de sua localização (MORIMOTO, 2010).

Utilizando recursos como GPS foi desenvolvido aplicativo para dispositivos móveis Android para o registro de rotas realizadas pelo usuário. A palavra rota é utilizada no sentido de um caminho percorrido pelo usuário não interessando o meio, embora a ideia inicial tenha sido para caminhos no estilo trilhas realizados a pé. Uma versão inicial deste aplicativo foi desenvolvida pelo autor deste trabalho (MACHADO, 2017). Neste trabalho este aplicativo foi complementado e foi desenvolvida uma aplicação *web* para o registro, o compartilhamento e a consulta dessas rotas.

As rotas armazenadas no dispositivo móvel do usuário poderão ser cadastradas a partir de *upload* de dados relacionados à rota que estão armazenados no dispositivo móvel do usuário e de informações complementares adicionadas quando do cadastro na aplicação *web*. Essas rotas poderão ser consultadas e baixadas por outros usuários. Visando, assim, que pessoas que realizam trilhas, por exemplo, possam saber de caminhos percorridos por outras pessoas, obter dados antes de realizar uma determinada rota (inclusive pelas imagens que podem ser armazenadas) e comentar sobre uma rota e outros.

1.2 OBJETIVOS

A seguir são apresentados o objetivo geral e os objetivos específicos alcançados com a realização deste trabalho.

1.2.1 Objetivo Geral

Desenvolver uma aplicação *web* para gerenciamento das rotas percorridas, registradas por meio de um aplicativo móvel, a interação e o compartilhamento de rotas e de opiniões dos usuários.

1.2.2 Objetivos Específicos

- Complementar requisitos em um aplicativo móvel para a marcação de pontos registrando uma rota realizada pelo usuário.
- Permitir que o usuário possa refazer uma rota, com a indicação visual de distância/proximidade do usuário dessa rota por meio de traçados.
- Possibilitar que o usuário possa configurar a distância ou o tempo para marcação dos pontos da rota.
- Obter a distância da rota por meio do cálculo da distância entre os pontos que foram marcados e que compõem uma rota.
- Permitir que o usuário possa efetuar autenticação por meio de uma conta vinculada ao Google.
- Permitir que o usuário possa capturar imagens e integrá-las à rota percorrida.
- Permitir que o usuário possa gerenciar suas rotas em na aplicação *web*.

- Permitir que outros usuários possam visualizar e baixar rotas percorridas por outros usuários.
- Permitir adicionar comentários em uma rota, possibilitando, assim, que os usuários possam expressar sua opinião sobre determinado trajeto e evidenciar dificuldades ou atrativos que a rota possa oferecer.

1.3 JUSTIFICATIVA

A marcação de rotas pode ser vista com duas funcionalidades básicas quando para uso não vinculado a negócio. Uma delas é a definição de um caminho que possa ser posteriormente percorrido. Por exemplo: marcar uma rota ou caminho de uma trilha desconhecida, auxiliando no retorno ou que outras pessoas possam utilizar essas marcações para realizar a mesma rota. Outra aplicação básica é medir a distância entre o início e o final da rota percorrida, por exemplo: uma pessoa quer saber a distância que andou a pé, de bicicleta ou de carro em um determinado trajeto.

A definição de rotas pode ser bastante útil para pessoas que se deslocam em locais com trilhas ou estradas não definidas, não devidamente sinalizadas, desconhecidas ou de acesso e localização difíceis, como matas, pântanos, áreas desérticas ou até em uma cidade ou local desconhecido. As marcações nas rotas, definidas quando o caminho foi realizado, podem ser utilizadas para o retorno. Essas rotas também podem ser utilizadas pela própria pessoa ou por outras que desejam fazer (refazer) o mesmo trajeto. A adição de imagens à rota facilita a identificação de pontos, evidenciando possíveis dificuldades ou, destacando locais na paisagem que sejam de observação interessante, como, por exemplo, uma cachoeira, um rochedo ou uma árvore.

Embora a justificativa essencial da aplicabilidade prática na aplicação *web* e o aplicativo *mobile* desenvolvidos por meio deste trabalho sejam para que pessoas possam compartilhar dados de trajetos percorridos, no sentido, de motivar atividades como *trakking*, caminhadas e outras relacionadas ao ecoturismo, por exemplo; em termos de aplicações comerciais são diversos os usos de mapeamento de rotas, por exemplo, para empresas de logística, entrega e transportadoras.

1.4 ESTRUTURA DO TRABALHO

Este trabalho, para um melhor acompanhamento e entendimento, está organizado em capítulos. No Capítulo 2 são apresentadas as tecnologias, as ferramentas e as principais atividades realizadas no desenvolvimento do trabalho. No Capítulo 3 é apresentada a aplicação *web* desenvolvida, também são apresentadas funcionalidades do aplicativo *mobile* (as pré-existentes, para a compreensão da regra de negócio do aplicativo e as desenvolvidas como resultado deste trabalho). Por fim estão as considerações finais seguidas das referências utilizadas no texto.

2 FERRAMENTAS, TECNOLOGIAS E PROCEDIMENTOS

A seguir estão os materiais e o método utilizados para a modelagem e a implementação obtidos como resultado da realização deste trabalho.

2.1 FERRAMENTAS E TECNOLOGIAS

No Quadro 1 estão as ferramentas e as tecnologias utilizadas na análise e no desenvolvimento da aplicação *mobile*.

Quadro 1 - Ferramentas e tecnologias utilizadas aplicativo *mobile*

Ferramenta / Tecnologia	**Versão**	**Referência (site)**	**Finalidade**
Google Maps API	24.0.2	https://developers.google.com/maps/documentation/android-api/?hl=pt-br	*Application Programming Interface* (API) de integração com mapas.
Google Goecoder API V2	1.2.2	https://developers.google.com/maps/documentation/geocoding/intro?hl=pt-br	Captura de descrição de localização por meio de coordenadas.
Google play service authentication	16.0.1	https://developers.google.com/identity/sign-in/android/start-integrating	API de autenticação com contas Google.
Android Studio	2.0	https://developer.android.com/studio/index.html?hl=pt-br	Ferramenta de desenvolvimento Android.
Android SDK	2.0	http://developer.android.com/sdk	*Kit* de desenvolvimento de *software* do Android.
SQLite	3.19	https://www.sqlite.org/	Armazenamento de dados no dispositivo móvel.
Spring Rest Template	1.0	https://projects.spring.io/spring-android/	*Framework* de desenvolvimento de integração com *web services*.
Fasterxml Jackson core	2.3	https://github.com/FasterXML/jackson	Framework de Streaming, Annotation, Databind Json.

Fonte: autoria própria.

No Quadro 2 estão as ferramentas e as tecnologias utilizadas na análise e no desenvolvimento da aplicação *web* e dos *services* de integração.

Quadro 2 – Ferramentas e tecnologias aplicação *web*

Ferramenta / Tecnologia	**Versão**	**Referência (site)**	**Finalidade**
Visual Studio Code	24.0.2	https://code.visualstudio.com/	Ferramenta de desenvolvimento *web*.
Angular	8.0.1	https://angular.io/	*Framework* Google de contrução de aplicações *web*
Primeng	8.0.1	https://www.primefaces.org/primeng/	Coleção de componentes para desenvolvimento web.
Java Script	1.8.5	https://www.javascript.com/	Linguagem de programação *web*.
Eclipse Oxigen.3 Java EE	4.7.3a	https://www.eclipse.org	Ferramenta de desenvolvimento do servidor de *backend* da aplicacao *web* e *mobile.*
Spring Boot	2.2.1	https://spring.io/projects/spring-boot	*Framework* de configurações para aplicações Spring
Spring Security	5.2.1	https://spring.io/projects/spring-security	*Framework* de autenticação e segurança de aplicações
Spring Data JPA	2.2.2	https://spring.io/projects/spring-data-jpa	*Framework* auxiliar para criação de repositório de percistência de dados

Fonte: autoria própria.

2.2 PROCEDIMENTOS TÉCNICOS

O método consiste nas atividades de levantamento de requisitos, modelagem desses requisitos e na implementação do aplicativo móvel e da aplicação *web*. A seguir está descrita sucintamente a realização dessas etapas.

a) **Levantamento de requisitos**

A ideia da implementação do aplicativo surgiu a partir do desenvolvimento de outra aplicação para monitoramento de localização, pelo autor deste trabalho (MACHADO, 2017). No desenvolvimento dessa aplicação verificou-se a utilidade de uma ferramenta para armazenar dados de caminhos (rotas) percorridos, registrar informações de tal caminho por meio de imagens (fotos) e possibilitar refazer esses caminhos com o auxílio de indicação se a rota está ou não sendo seguida. Posteriormente, viu-se a possibilidade de desenvolver uma

pequena rede social, por meio da qual os usuários do aplicativo poderiam disponibilizar suas informações para que outras pessoas tivessem conhecimento delas.

As funcionalidades foram desenvolvidas a partir da análise de como as pessoas ficam confusas (como vendedores em uma cidade nova) ao visitar locais desconhecidos e não saberem exatamente como chegar a determinado local para poder voltar posteriormente, ou mesmo para saberem como efetuar o retorno para o local de origem. Outra motivação e circunstância de análise para definição dos requisitos foi a observação que pessoas acabam se perdendo em trilhas por não saberem o caminho de volta ou por ficar caminhando em círculos ou desorientadas em locais desconhecidos.

Para o levantamento de requisitos complementares e visando um viés prático para usuários não empresariais, foram consideradas indicações de funcionalidades que seriam úteis em uma aplicação desse tipo realizadas pela Profa. Beatriz T. Borsoi e pelo Prof. Robison C. Brito.

b) **Análise e projeto**

A análise e o projeto consistiram na modelagem dos requisitos. A modelagem dos requisitos é simples em termos de casos de uso e de persistência de dados. O aplicativo ficará armazenado no dispositivo móvel do usuário e não haverá necessidade de controle de permissões de acesso ou de restrição de acesso às funcionalidades, até porque haverá apenas um perfil ativo com acesso total aos dados do usuário e as diversas telas do aplicativo. A autenticação será necessária, apenas para identificar a qual usuário pertence determinada rota, para que na aplicação web seja possível a distinção entre as mesmas. O banco de dados do aplicativo *mobile* consiste em apenas duas tabelas para persistência das rotas (os caminhos percorridos) e uma para registro de informações do usuário autenticado. Toda a parte de controle de *login*, autenticação, redefinição de senhas e alteração de informações relacionadas ao usuário foram delegadas a *Application Programming Interface* (API) do Google Authentication, sendo que para utilizar o aplicativo, tanto *web* como *mobile*, se faz necessário ter uma conta Google.

Já o servidor contará com uma réplica das tabelas presentes no aplicativo móvel, adicionado três novas tabelas. Uma para armazenamento dos dados de usuários autenticados via Google na aplicação web. Outra para comentários realizados pelos usuários nas rotas disponíveis no aplicativo *web*. E uma terceira tabela para armazenamento temporário sendo salvas as rotas que um usuário específico deseja fazer *download* para seu dispositivo.

c) Implementação

A implementação da aplicação foi dividida em três etapas, sendo cada uma delas dividida em subetapas: aplicativo *mobile*, aplicação *web* e integração de dados entre aplicações.

- **Aplicativo *Mobile***

A primeira etapa focou no desenvolvimento do aplicativo móvel e essa foi subdividida em nove etapas:

1) Tela inicial e preferências - a etapa inicial de implementação foi focada no desenvolvimento da tela de abertura do aplicativo (*splash*), da tela de menu de opções, com uso de *adapter* próprio, na implementação de listas personalizada de itens e de *dialogs*.

2) Desenvolvimento de tela de autenticação de usuário e sua regra de negócio - o desenvolvimento da tela de autenticação de usuário (*login*) ficou resumida a um botão simples que ao ser 'clicado' consome a API de autenticação do Google, recebendo ao final um objeto com dados do usuário autenticado. Ainda nessa tela há informações de política de privacidade, alertando o usuário sobre os possíveis riscos legais da utilização do aplicativo, ficando a este a escolha em acatar possíveis responsabilidades.

3) Captura de posições GPS e apresentação em mapa - etapa com foco no desenvolvimento de rotina de captura de sinal GPS por meio da API Google Maps com o processamento dos dados recebidos e exibição em tela de mapa no dispositivo Android. Ainda na tela de mapa foi implementado botões, sendo eles para facilitar a aplicação de *zoom*, captura de imagens por meio da câmera do dispositivo e re-percurso de uma rota com a ajuda do GPS para indicar a posição atual em relação a uma rota já percorrida naquele local, que servira como guia.

4) Desenvolvimento de tabelas SQLite para persistência - a quarta etapa foi voltada para a projeção das tabelas necessárias, buscando suprir as necessidades de persistências da aplicação. Foram projetadas três tabelas, duas para armazenamento das rotas capturadas no passo três, sendo uma delas com chave estrangeira buscando maior integridade e referenciamento de registros de rotas salvas, e uma para armazenar informações do usuário conectado, dados estes, recuperados no passo dois deste sequencial e que, em próximas versões possam ser úteis.

5) Identificação e persistência de rotas - nessa etapa foi adaptada a tela do aplicativo para, solicitar o nome e meio utilizado para percurso da rota, ao iniciar o rastreamento, também foi desenvolvida a classe *Data Access Object* (DAO) para persistência e recuperação

de dados da base SQLite, ainda foi desenvolvida uma classe *adapter* para ajustes na persistência dos dados e que, posteriormente, será utilizada, na integração com o *web service*.

6) Listagem de rotas salvas - nesta etapa foi desenvolvido um novo *adapter* para a tela de listagem para exibir as rotas salvas no dispositivo *mobile*, disponibilizando opção para o usuário excluir uma rota ou visualizar a rota percorrida em uma tela de mapa.

7) Serviços de captura de posição GPS e busca de localização - nessa etapa foi integrada a rotina de captura de localização para função de serviço do Android, possibilitando iniciá-lo de outras telas. Além disso, foi inserido um *timmer* nesse serviço, possibilitando maior precisão no controle do tempo de execução da captura dos pontos da rota. Também foi implementado um serviço que percorre as posições persistidas na base SQLite, calculando as distâncias e buscando a descrição de cada ponto salvo utilizando a API Google Geocoder.

8) Funcionalidade para refazer uma rota - nessa etapa foi implementada opção para o usuário realizar novamente uma rota. Essa opção está disponível ao visualizar uma rota específica, no mapa, e pode ser acessada pelo botão 'Guiar'. Ao selecioná-lo, a posição atual do usuário começará a ser exibida na tela de mapa com marcadores distintos dos da rota base. Assim, o usuário poderá se orientar em qual sentido se deslocar para chegar a um determinado ponto da rota base.

9) Integração com *web service* - nessa última etapa foram criadas classes para coletar, organizar, transformar e transmitir uma lista de coordenadas, imagens e informações no formato Json para um *web service* que disponibilizará tais informações em uma aplicação *web*. Também foi ajustada a listagem de rotas, desenvolvida no passo 6, com a adição de um novo botão que possibilita a publicação da rota para os demais usuários.

- **Aplicação Web**

A segunda etapa focou no desenvolvimento das telas *web*. Essa também foi subdividida em 6 subetapas.

1) Desenvolvimento das funcionalidades de autenticação - desenvolvidas as funcionalidades referentes ao *login* e à autenticação de navegação, utilizando a API Google Autenticador para efetuar o *login*, combinando ainda a API Spring Security que fará a autenticação de navegação e controle de sessão dentro da página *web*. Dessa forma, a cada *Uniform Resource Locator* (URL) acessada pelo usuário, o Spring validará se o *login* ainda é válido e caso o resultado seja negativo, redirecionará para a página de *login* da aplicação. Da mesma forma, que no aplicativo, toda a parte de autenticação de usuário e senha, bem como possível recuperação ou redefinição de senha, é delegada à API do Google. Restando apenas à

aplicação consumir a API e ao final recuperar um objeto contendo os dados do usuário autenticado, que serão salvos em banco de dados para possível utilização posterior, ou a rejeição do *login*, redirecionando novamente para a tela de autenticação apresentando erro.

2) Listagem das rotas do usuário autenticado e listagem das rotas dos demais usuários - após a autenticação efetuada, o usuário é direcionado para a tela de listagem de suas rotas publicadas. Nessa tela ele terá a opção de excluir, baixar ou visualizar uma determinada rota de sua autoria. Também, semelhante a essa listagem, existe a listagem das rotas dos demais usuários, sendo possível ao usuário visualizar e baixar a rota de terceiros para o seu aplicativo, caso seja de seu interesse.

3) Visualização de uma rota no mapa - a visualização das rotas em mapa, faz uso também da API do Google Maps, após recebida a lista dos pontos de uma referida rota, a aplicação, utilizando diretivas do Angular, cria na tela a representação gráfica da rota percorrida, exibindo os marcadores nas posições em que existem imagens que podem ser visualizadas com um clique.

4) Inserção e manutenção de comentários em uma rota - ainda na página do mapa, da etapa anterior, na parte inferior dessa, é disponibilizada uma área na qual os usuários podem deixar sua opinião a respeito da rota visualizada, no formato de comentário.

5) *Download* de rota para o aplicativo *mobile* - tanto na listagem de rotas próprias, como na listagem de rotas de outros usuários, foi adicionado um botão para que o usuário possa fazer *download* de uma determinada rota para o aplicativo. Ao clicar no botão é agendado um evento de *download*, que posteriormente será requisitado pela aplicação *mobile* ao servidor de integração e este empacotará todas as rotas a serem baixadas em um único arquivo Json, no mesmo formato utilizado pelo aplicativo *mobile*, na etapa 9 de desenvolvimento anterior.

6) Exclusão de rotas - a exclusão de rotas só é possível para rotas do próprio usuário. Essa operação é realizada por meio de um botão disponível na listagem de rotas próprias, sendo que ao ser clicado, a rota é removida da base de dados *web*, ficando disponível caso o usuário a tenha baixado e ainda esteja em seu aplicativo. A rota não e removida do aplicativo de outros usuários que a tenham baixado.

- **Integração de dados entre aplicações**

A terceira fase focou na integração entre as duas primeiras etapas utilizando serviços *web* seguindo o modelo *Representational State Transfer* (REST) de comunicação. Essa etapa se dividiu em 3 etapas que são:

1) ‘Porta’ de comunicação com aplicativo *mobile* - neste módulo foi desenvolvido um único serviço responsável por receber um arquivo Json, processá-lo e devolvê-lo ao remetente. Todo o processo é feito de modo síncrono, ficando o cliente conectado aguardando o retorno da resposta do processamento.

2) ‘Porta’ de autenticação e comunicação com aplicativo *web* - nessa segunda parte do serviço REST foi implementado um segundo módulo, com apenas dois serviço. Sendo um deles responsável pela parte de autenticação do usuário e o outro responsável pela comunicação dos dados de navegação com a aplicação *web*, sendo que este recebe um arquivo Json na requisição, processa-o e o devolve como resposta ao cliente. Da mesma forma que na integração *mobile*, os processos são realizados de modo síncrono.

3) Persistência dos dados em banco Postgresql - na terceira etapa de desenvolvimento dos serviços de integração, foram projetadas as tabelas para persistência dos dados recebidos, tanto do aplicativo *mobile*, como rotas percorridas, quanto da aplicação *web*, como dados de usuários autenticados e os comentários.

3 RESULTADO

Este capítulo apresenta o resultado da realização deste trabalho.

3.1 ESCOPO DO SISTEMA

O aplicativo desenvolvido tem a finalidade principal de ser um rastreador que armazenará rotas percorridas pelo usuário, recuperando, posteriormente, posições marcadas na rota, horário de ocorrência das marcações, locais de marcação (pontos geográficos que definem a posição no mapa físico com marcador), velocidade aproximada de deslocamento e distância percorrida, bem como de fotografias e comentários armazenados vinculados à rota. A cada intervalo de pontos marcados pelo usuário, o aplicativo medirá a distância e, assim, ao final do trajeto será obtida por somatório a distância total, ou somatórias parciais de acordo com o avanço do deslocamento.

O aplicativo permite ao usuário refazer uma rota já realizada, ou seja, o aplicativo indicará, por meio dos pontos projetados no mapa, a direção que o usuário deverá seguir para realizar o caminho indicado pela rota. Através da rota desenhada no mapa e do ponto da posição atual do usuário o mesmo poderá, consultando o mapa, saber em qual direção seguir para que alcance seu objetivo.

O aplicativo permite ainda a captura de imagens anexadas à rota e podem ser visualizadas a qualquer momento já que são imediatamente exibidas no mapa, facilitando a identificação do local onde o usuário está e possibilitando uma melhor orientação em caminhadas posteriores. Essas imagens podem servir também para que outros usuários conheçam a rota por meio de imagens, por exemplo, cachoeiras, rochedos e outros. Servindo, assim, como forma de divulgação da referida rota e como pontos de localização.

O sistema desenvolvido (aplicação *web* e complementos no aplicativo *mobile*) como resultado deste trabalho, permitirá que o usuário armazene todos os caminhos pelos quais passou, auxiliando-a a refazer o caminho ou retornar ao ponto de partida evitando errar o caminho. Como exemplos de utilização dessa aplicação podem-se citar: caminhadas, trilhas ecológicas, trajetórias em cidades ou bairros desconhecidos e até com registros de onde veículos de entrega de uma empresa passaram, por exemplo. O sistema possibilita que tais percursos possam ser compartilhados com outros usuários no formato de uma rede social, que pode ser acessada em uma página *web*, utilizando o mesmo usuário e senha de acesso ao

aplicativo. Assim, os usuários podem interagir e baixar essas rotas para seus aplicativos ou ainda deixar comentários em uma determinada rota, expressando suas opiniões sobre tal. A solução proposta considera o contexto apresentado a seguir:

Um usuário pode percorrer mais de um caminho e manter registros separados desses caminhos para posterior deslocamento de retorno. O aplicativo funciona mesmo sem sinal de internet (WiFi) e sem estar em conexão com a rede de operadora. Contudo, o dispositivo deve ter módulo GPS (que é presente na maioria dos dispositivos móveis), também é necessário, para um melhor desempenho e precisão, que o usuário tenha boa visada do céu para que o sistema GPS obtenha os sinais dos satélites de triangulação de posição. Também, para efetuar autenticação pela primeira vez no aplicativo, é necessário que o usuário possua acesso à rede de dados (internet) e possua uma conta de *e-mail* cadastrada nos servidores do Google Mail ou Gmail. Ainda, para utilização das funções de compartilhamento e *download* de rotas, também se faz necessário o acesso à rede de dados.

Ao abrir a aplicação será efetuada uma carga do último *login* acessado e caso nenhum *login* esteja salvo, será exibida a tela de autenticação na aplicação *web*, com *link* para efetuar o cadastro.

Após o usuário efetuar o *login*, utilizando sua conta de *email*, será iniciado um serviço de varredura na base de dados dos pontos armazenados que estejam sem informação de descrição bem como a tentativa de conexão com a base de dados Google Maps, para a recuperação das descrições dos pontos percorridos. Também será efetuado o recálculo, por meio de uma API, das distâncias de cada rota armazenada na base de dados local do aplicativo, que ainda não tenham sido recalculadas.

Após aberta a aplicação, um menu de opções é apresentado ao usuário que poderá iniciar uma nova rota, continuar a última rota percorrida, visualizar todas as suas rotas, visualizar todas as rotas baixadas, visualizar, em um mapa, a última rota gravada, acessar um menu de configurações ou fazer *logoff* do aplicativo.

Nas opções de visualização de suas rotas ou de rotas baixadas, será exibida uma lista de rotas e botões para exclusão ou visualização. Ainda nas rotas de sua propriedade (salvas pelo usuário), haverá um botão possibilitando que ele compartilhe a rota desejada e uma página *web*, na qual os demais usuários poderão visualizá-la e baixá-la, caso seja de seu interesse, além de deixar comentários.

As rotas serão armazenadas em base SQLite no próprio dispositivo móvel para que elas possam ser recuperadas posteriormente. A recuperação se refere à funcionalidade de

repercorrer uma rota ou de consultar dados que foram armazenados, além da opção de compartilhamento com os demais usuários.

O usuário também pode excluir uma rota de seu aplicativo caso desejar, seja ela própria ou baixada de outro usuário. Na página *web*, a opção de exclusão de rota só é disponível para rotas da própria autoria do usuário, sendo essa informação revalidada no momento da exclusão no servidor para evitar possíveis fraudes no sistema.

Na aplicação *web*, como mencionado, também é possível que os usuários deixem opiniões em uma rota, sendo que esses comentários podem ser alterados ou excluídos pelo próprio autor. Em primeira versão, não haverá a funcionalidade de moderação para tais comentários. O usuário utilizador do aplicativo e autor do comentário ficará inteiramente responsável por possíveis comentários impróprios, ofensivos ou outros que possam ferir direitos ou imagem de outros usuários ou terceiros.

3.2 MODELAGEM DO SISTEMA

Para uma melhor apresentação, a modelagem foi dividida em três partes, representando cada uma, uma aplicação distinta no sistema como um todo, sendo elas: aplicativo *mobile*, aplicação *web* e *web services* de integração.

3.2.1 Modelagem do Aplicativo *Mobile*

O Quadro 3 apresenta a listagem dos requisitos funcionais (RF) identificados para o aplicativo *mobile*.

Quadro 3 – Requisitos funcionais do aplicativo *mobile*

Identificação	Nome	Descrição
RF01	Efetuar login	É solicitado ao usuário autenticação, realizada via Google Auth, para poder utilizar o aplicativo. Após autenticação aceita e o usuário concordar com os termos de uso, o aplicativo é liberado para uso.
RF02	Armazenar rota	É solicitado um nome para a rota e posteriormente iniciado um serviço de captura de localização gravando em base de dados SQLite.
RF03	Ver rota atual	Exibe um mapa com os pontos já capturados e acrescenta novos pontos conforme o avanço no deslocamento.
RF04	Visualizar rotas	Exibe um mapa com todos os pontos de cada uma das rotas já percorridas ou de uma rota específica.

Rf05	Compartilhar rota	Publica a rota selecionada em uma página *web*. Nessa página os demais usuários poderão visualizar, baixar e comentar sobre as rotas.
RF06	Excluir dados	Exclui fisicamente uma rota específica e seus respectivos pontos e imagens armazenados e vinculados a ela.
RF07	Ver configurações	Exibe a tela na qual é possível executar várias configurações do aplicativo.
RF08	Rastrear *service*	Ao iniciar um rastreamento esse serviço é responsável pela rotina de captura dos pontos e chamada dos métodos de gravação no banco de dados.
RF09	Repercorrer rota	Escolher uma rota registrada e refazer a rota. O aplicativo vai indicando se o usuário está se deslocando fora da rota.
RF10	Fazer *logout*	Faz *logout* para possibilitar o uso por outro usuário.

Fonte: autoria própria.

A listagem que está no Quadro 4 apresenta os requisitos não funcionais (RNF) identificados para o aplicativo *mobile* como um todo, também denominados de requisitos suplementares. Os requisitos não funcionais explicitam regras de negócio, requisitos de qualidade, desempenho entre outros que se aplicam ao sistema como um todo ou estão vinculados a requisitos funcionais.

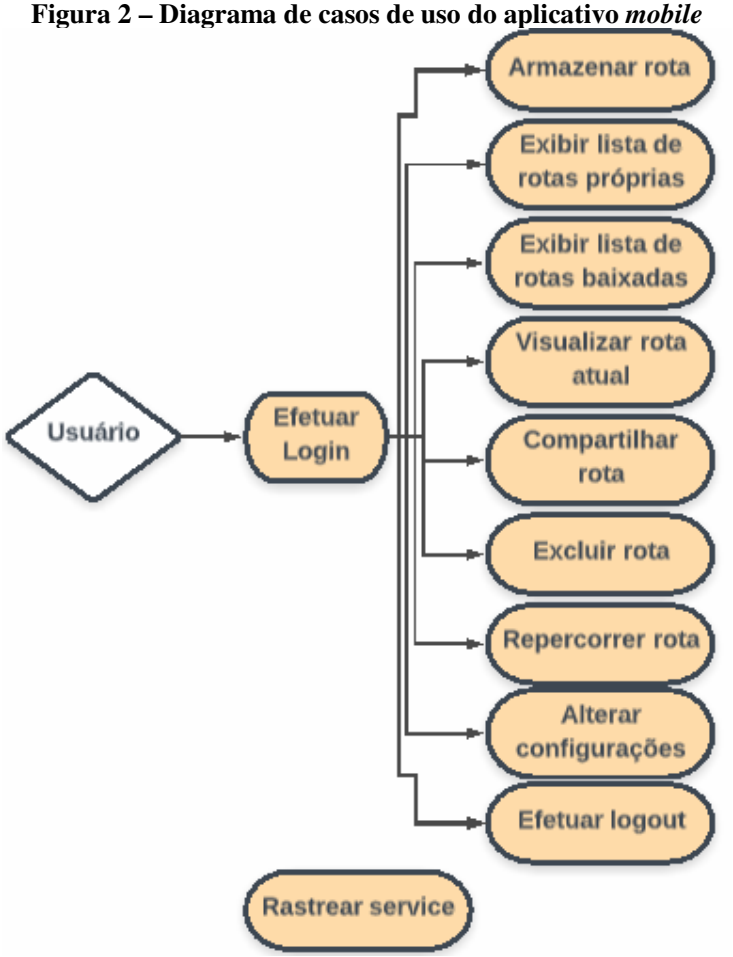
Quadro 4 – Requisitos não funcionais do aplicativo *mobile*

Identificação	**Nome**	**Descrição**
RNF01	Tela *splash*	Ao iniciar o aplicativo é exibida uma tela do tipo *splash*. Nessa tela é iniciado um serviço de atualização de cálculo de distância de rota bem como atualização das descrições dos pontos caso exista conexão com a Internet.
RNF02	Thread para gravar dados	*Thread* responsável pela persistência dos dados na base de dados, utilizada para evitar travamento da aplicação.
RNF03	Thread monitor	*Thread* responsável por, em intervalo de tempos, verificar se existem rotas no servidor para serem baixadas para o aplicativo.

Fonte: autoria própria.

O diagrama de casos de uso apresentado na Figura 2 contém as funcionalidades essenciais do aplicativo móvel realizadas pelos seus usuários e pelo próprio aplicativo em rotinas de disparo automático.

Figura 2 – Diagrama de casos de uso do aplicativo *mobile*



Fonte: autoria própria.

No Quadro 5 é descrito o caso de uso de *login* de usuário no aplicativo *mobile*.

Quadro 5 – Caso de uso efetuar login

Caso de uso: Efetuar *login*. **Descrição:** Esse caso de uso consome uma API de autenticação do Google que se responsabiliza em solicitar usuário e senha, validar os dados e retornar para a aplicação o resultado da autenticação, sendo um objeto com informações do usuário, caso a autenticação seja feita com sucesso, ou uma falha caso a autenticação não seja bem-sucedida. **Evento Iniciador:** Clicar no botão efetuar *login* na página inicial do aplicativo. **Atores:** Usuário **Pré-condição:** Caixa de seleção de aceite dos termos de uso selecionada, confirmando que o usuário leu os termos e aceita as responsabilidades. **Sequência de Eventos:** 1. Ator clica no botão de *login*.

2. API Google é chamada pelo aplicativo. 3. API solicita os dados do usuário para efetuar autenticação. 4. API retorna resposta da autenticação. **Pós-Condição:** 1. Dados da resposta são lidos e salvos em base SQLite. 2. Menu inicial da aplicação é aberto

Nome do fluxo alternativo (extensão)	Descrição
Linha 1: clicar no botão *login*	1.1 Se caixa de seleção de ciência e aceite dos termos de uso não está marcada, mensagem de advertência é exibida e a página inicial é reapresentada ao usuário.
Linha 4: retorno da resposta	4.1 Caso o usuário cancele a tentativa de *login*, uma falha de autenticação é exibida e a tela inicial de *login* é exibida.
Linha 1: Pós-Condição dados da resposta	1.1 Se ocorrer erro de autenticação, uma mensagem de alerta é exibida e a página inicial de *login* é reapresentada ao usuário.

Fonte: autoria própria.

No Quadro 6 está a descrição do caso de uso armazenar rota.

Quadro 6 – Caso de uso armazenar rota

Caso de uso: Armazenar rota. **Descrição**: Inicia processo de captura de coordenadas de localização e efetua a persistência dos dados em banco de dados. **Evento Iniciador**: Clicar no botão iniciar ou continuar rota. **Atores**: Usuário. **Pré-condição**: Ter sido inserido um nome e um tipo para a nova rota ou ter sido selecionado a opção continuar para continuar a ultima rota. Sensor GPS do dispositivo estar ativo ou dispositivo com conexão à rede de dados móveis. **Sequência de Eventos**: 1. Iniciado um serviço que será responsável por gerenciar a captura dos dados. 2. Sistema verifica se já está sendo feita a captura dos dados. 3. Sistema verifica se o sensor GPS está ativo ou se há conectividade de dados móveis. 5. Sistema inicia a captura de pontos por meio da API do Google Maps no qual é iniciado um método assíncrono que verifica a posição atual. 6. Quando um novo ponto é encontrado, um objeto *location* é retornado do método de captura. Esse objeto contém vários dados como posição, altitude, velocidade dentre outros. Esses dados são recuperados e uma nova *thread* é iniciada para persistir os dados, sendo o ciclo de captura repetido. **Pós-Condição**: Dados do ponto capturado inseridos no banco de dados.

Nome do fluxo alternativo (extensão)	Descrição
Linha 2: verificação de sistema ativo	2.1 Se já estiver ativo o rastreamento, então o sistema desativa a captura dos dados de GPS e retorna para a tela principal da aplicação.

Linha 3: verificação de sensor GPS	3.1. Se o sensor GPS estiver inativo e nenhuma rede móvel conectada, então sistema exibe mensagem informativa e abre tela de configurações do dispositivo para usuário ativar o sistema de GPS. 3.2 Se sensor não for ativo, sistema aborta operação e retorna para o menu principal.

Fonte: autoria própria.

O caso de uso visualizar rota atual é apresentado no Quadro 7.

Quadro 7 – Caso de uso visualizar rota atual

Caso de uso:
Visualizar rota atual.
Descrição:
Esse caso de uso exibe um mapa com os pontos já percorridos da rota atual ou da rota selecionada, se a captura de novos pontos estiver ativa então a opção de repercorrer a rota e desabilitada no mapa e caso novos pontos sejam capturados o mapa é automaticamente atualizado. Caso o rastreio não esteja ativo, o usuário terá a opção de repercorrer a rota exibida por meio de um botão no mapa.
Evento Iniciador:
Escolha da opção correspondente à ação visualizar rota atual.
Atores:
Usuário
Pré-condição:
Existir rota(s) gravada(s) ou que tenha sido selecionada pelo usuário.
Sequência de Eventos:
1. Ator seleciona opção para visualizar rota atual.
2. Sistema inicia nova tela com componente de mapa.
3. Sistema recupera da base de dados os pontos referentes à rota a ser exibida.
4. Sistema insere no mapa os marcadores de cada ponto.
5. Sistema insere traçado entre os pontos.
6. Sistema verifica se rastreador está ativo.

Pós-Condição: não há.

Nome do fluxo alternativo (extensão)	Descrição
Linha 6: ver rastreador	6.1 Se rastreador estiver ativo, então sistema inicia rotina de sincronização e inserção de novos pontos ao mapa.

Fonte: autoria própria.

O caso de uso exibir lista de rotas é apresentado no Quadro 8. O mesmo caso de uso descreve tanto a lista de rotas próprias, quanto à lista de rotas baixadas.

Quadro 8 – Caso de uso exibir lista de rotas

Caso de uso:
Exibir lista de rotas.
Descrição:
Apresentar a lista de rotas cadastradas necessárias para que um percurso de rota seja refeito.
Evento Iniciador:
Escolha da opção ver lista de rotas no menu principal.

Atores:
Usuário
Pré-condição:
Existir rota(s) gravada(s).
Sequência de Eventos:
1. Ator seleciona opção de visualizar lista de rotas.
2. Sistema busca na base de dados todas as rotas salvas.
3. Sistema exibe mensagem caso nenhuma rota seja encontrada.
4. Sistema inicia nova tela de listagem com a lista das rotas recuperadas.
5. Sistema apresenta opção de excluir uma rota da lista.
6. Sistema apresenta opção de visualizar em um mapa a rota da lista.
7. Sistema apresenta opção de compartilhamento da rota.
Pós-Condição:
Operação de visualização ou exclusão de uma rota.

Nome do fluxo alternativo (extensão)	Descrição
Linha 6: ver Mapa	6.1 Ao clicar em ver rota é exibida nova tela com um componente de mapa sendo são inseridos todos os pontos da rota selecionada.
Linha 5: exclusão de registro de rota.	5.1 Se solicita exclusão de registro, o sistema chama o método de exclusão passando via parâmetro o ID da rota selecionada.
Linha 7: compartilhamento de rota	7.1 A opção de compartilhamento de rota só esta disponível para rotas da própria autoria do usuário.

Fonte: autoria própria.

O caso de uso compartilhar rota é apresentado no Quadro 9.

Quadro 9 – Caso de uso compartilhar rota

Caso de uso:
Compartilhar rota.
Descrição:
Operação de compartilhamento de uma rota em página *web*, com os demais usuários da aplicação.
Evento Iniciador:
Escolha da opção para compartilhar dados.
Atores:
Usuário
Pré-condição:
Existir rota selecionada.
Sequência de Eventos:
1. Ator seleciona opção de compartilhar rota.
2. Sistema executa *thread* paralela de comunicação para enviar rota selecionada para o servidor.
3. Tela de lista de rotas é exibida novamente.
Pós-Condição:
1. Thread é executada buscando a rota selecionada, seus pontos e imagens na base SQLite.
2. Dados são adicionados a um arquivo tipo JSON.
3. Arquivo é enviado ao servidor de forma síncrona.
4. Resposta do servidor é recebida.
5. Resposta é lida e "id" dos dados gravados no servidor *web* é vinculado aos pontos de rota

que foram gravados com sucesso no servidor *web*.
6. *Thread* é encerrada.

Nome do fluxo alternativo (extensão)	Descrição
Linha 1: Pós-condição.	1.1 Caso nenhuma informação da rota seja localizada, passo 6 é executado.

Fonte: autoria própria.

No Quadro 10 é apresentado o caso de uso excluir rota.

Quadro 10 – Caso de uso excluir rota

Caso de uso:
Excluir rota.
Descrição:
Exclui a rota selecionada do aplicativo.
Evento Iniciador:
Usuário selecionar opção excluir na lista de rotas.
Atores:
Usuário.
Pré-condição:
Haver rotas próprias ou baixadas no aplicativo.
Sequência de Eventos:
1. Ator seleciona a rota.
2. Ator clica na opção excluir.
3. Sistema solicita confirmação de exclusão.
3. Sistema executa *thread* paralela de exclusão da rota.
4. Sistema confirma exclusão da rota.
5. Sistema atualiza lista de rotas.
Pós-Condição:
Não há.

Nome do fluxo alternativo (extensão)	Descrição
Linha 3: Confirmação de exclusão.	3.1 Caso não seja confirmada a exclusão da rota, vai para o passo 5.

Fonte: autoria própria.

O caso de uso visualizar configurações é apresentado no Quadro 11.

Quadro 11 – Caso de uso visualizar configurações

Caso de uso:
Alterar configurações.
Descrição:
Exibição da tela de configurações do aplicativo, na qual o usuário pode efetuar parametrizações de como o aplicativo deve comportar-se para a captura dos dados de uma rota e exibição da rota no mapa.
Evento Iniciador:
Usuário selecionar opção de visualizar configurações no menu principal ou no menu suspenso.
Atores:
Usuário

Pré-condição:
Não há.
Sequência de Eventos:
1. Ator seleciona opção de visualizar configurações.
2. Sistema exibe tela de configurações.
3. Ator seleciona parâmetro a ser ajustado/alterado.
4. Sistema habilita campo para que o valor possa ser alterado.
5. Ator altera e solicita salvamento dos dados.
5. Sistema salva dados de preferência para posterior utilização.
Pós-Condição:
Não há.

Fonte: autoria própria.

O caso de uso rastrear *service* é apresentado no Quadro 12.

Quadro 12 – Caso de uso rastrear service

Caso de uso:
Rastrear *service*.
Descrição:
Serviço interno do sistema, responsável pela captura dos pontos de localização e persistência dos pontos recuperados em base de dados.
Evento Iniciador:
Caso de uso armazenar rota.
Atores:
Sistema.
Pré-condição:
Usuário solicitar início de captura de rota.
Sequência de Eventos:
1. Sistema instancia objeto LocationManager.
2. Sistema verifica disponibilidade de captura de dados de GPS.
3. Sistema inicia escuta de serviço de captura de novos pontos.
4. Resposta recebida na escuta.
5. Sistema valida dados retornados na escuta.
6. Sistema gera objeto de ponto de rota.
7. Sistema instancia rotina de persistência dos dados do objeto na base de dados.
Pós-Condição:
Sistema bloqueia na escuta enquanto novos dados não sejam retornados.

Fonte: autoria própria.

O caso de uso repercorrer (refazer) rota é apresentado no Quadro 13.

Quadro 13 – Caso de uso repercorrer rota

Caso de uso:
Repercorrer rota.
Descrição:
Quando um mapa é exibido apresentando os pontos de uma rota já percorrida e o rastreamento de novos pontos não esteja em execução, é exibido um botão na tela do mapa com a descrição guiar. Quando essa opção é selecionada, uma nova rotina da captura do ponto atual de localização do usuário

é iniciada e a cada ponto capturado é inserido no mapa um marcador diferenciado com a posição atual do usuário, bem como são exibidos os pontos da rota original para servirem de referência para que o usuário saiba para aonde deve se deslocar.

Ao clicar em um dos marcadores são exibidas informações da localização como data e hora, descrição da localização, velocidade e distância percorrida.

Evento Iniciador:

Ator seleciona botão Guiar.

Atores:

Usuário.

Pré-condição:

Existir rota já percorrida e não existir nenhuma rota em percurso.

Sequência de Eventos:

1. Ator seleciona botão para refazer rota.
2. Sistema captura os pontos da rota armazenados no banco.
3. Sistema renderiza mapa na tela e insere os pontos recuperados da base de dados.
4. Sistema inicia processo assíncrono que captura o ponto atual do usuário de acordo com tempo ou distância de deslocamento configurado pelo usuário em tela de preferência e apresenta no mapa o ponto exato para visualização do usuário.
5. Ator seleciona ponto de localização e informações sobre o ponto são exibidas.

Pós-Condição:

Ator solicitar parada do processo.

Nome do fluxo alternativo (extensão)	Descrição
Linha 4: Processo assíncrono	4.1 Sistema executa esse processo indeterminadas vezes até que o usuário solicite a parada por clique no botão parar.

Fonte: autoria própria.

3.2.2 Modelagem Aplicação Web

O Quadro 14 apresenta a listagem dos requisitos funcionais identificados para a aplicação *web*.

Quadro 14 – Requisitos funcionais da aplicação *web*

Identificação	Nome	Descrição
RF01	Efetuar *login*	É solicitado ao usuário autenticação, via Google Auth, para poder utilizar o sistema. Após a autenticação ser aceita, o aplicativo é liberado para uso.
RF02	Listar rotas próprias e/ou de terceiros	Exibe uma listagem com todas as rotas disponíveis na nuvem. Se a opção listar minhas rotas for selecionada, o sistema exibirá as rotas que pertencem ao usuário autenticado e que foram previamente compartilhadas via aplicativo. Se outras rotas forem selecionadas, o sistema exibirá uma lista com todas as rotas compartilhadas pelos demais usuários da aplicação.
RF03	Visualizar rota	Exibe um mapa com todos os pontos da rota selecionada e possíveis imagens que tenham sido anexadas à rota. E logo abaixo desse mapa, são exibidos comentários publicados pelos usuários sobre a rota.
Rf04	Baixar rota	Ao ser selecionada essa opção, a rota será baixada para o aplicativo que o usuário estiver autenticado ou que se

		autentique na próxima vez. Todas as imagens vinculadas às rotas também serão baixadas.
RF05	Excluir rota	Exclui fisicamente uma rota específica e seus respectivos pontos, imagens e comentários vinculados. A exclusão ocorre apenas na nuvem, mas permanece disponível nos aplicativos que a tenham baixado. A exclusão somente está disponível para rotas de autoria do usuário autenticado.
RF06	Inserir comentário	Insere um novo comentário vinculado ao usuário publicador e a rota que está sendo visualizada.
RF07	Excluir comentário	Disponível apenas para o autor do comentário, exclui o comentário selecionado.
RF08	Efetuar *logout*	Faz *logout* no sistema para possibilitar o uso por outro usuário.

Fonte: autoria própria.

A listagem que está no Quadro 15 apresenta os requisitos não funcionais identificados para a aplicação *web*

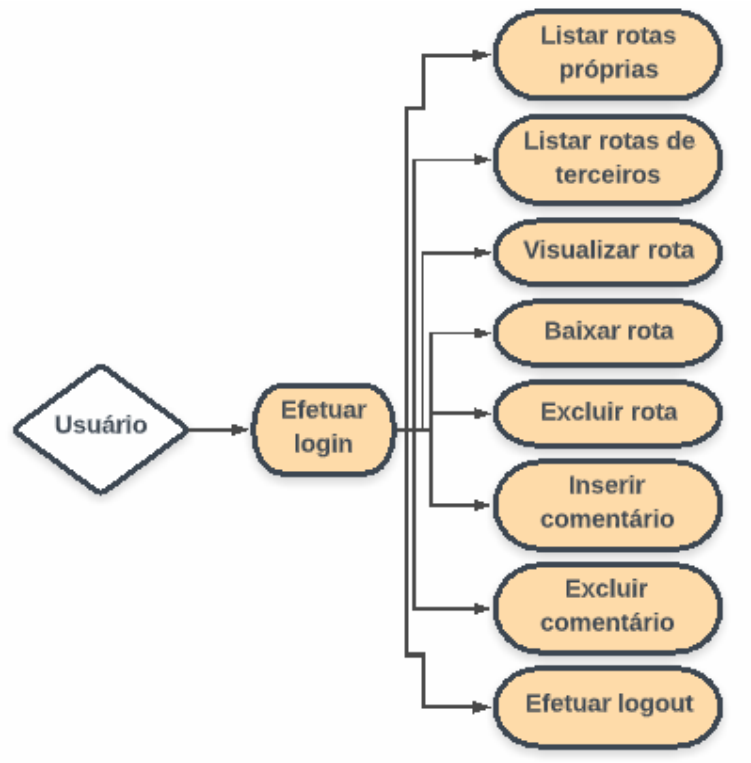
Quadro 15 – Requisitos não funcionais da aplicação *web*

Identificação	**Nome**	**Descrição**
RNF01	Thread *download*	*Thread* responsável por agendar um evento de *download* para um determinado usuário.

Fonte: autoria própria.

O diagrama de casos de uso apresentado na Figura 3 contém as funcionalidades essenciais do sistema realizadas pelos seus usuários e pela aplicação *web* em rotinas de disparo automático.

Figura 3 – Diagrama de casos de uso da aplicação *web*



Fonte: autoria própria.

No Quadro 16 é descrito o caso de uso de *login* de usuário no aplicativo *web*.

Quadro 16 – Caso de uso efetuar *login*

Caso de uso: Efetuar *login*. **Descrição:** Esse caso de uso consome uma API de autenticação do Google, a qual se responsabiliza em solicitar usuário e senha, validar os dados e retornar para a aplicação o resultado da autenticação, sendo um objeto com informações do usuário, caso a autenticação seja feita com sucesso, ou uma falha caso a autenticação não seja bem-sucedida. **Evento Iniciador:** Clicar no botão efetuar *login* na página inicial da aplicação. **Atores:** Usuário **Pré-condição:** Não há. **Sequência de Eventos:** 1. Ator clica no botão de *login*. 2. API Google é chamada pelo aplicativo. 3. API solicita os dados do usuário para efetuar autenticação. 4. API retorna resposta da autenticação. **Pós-Condição:** 1. Dados da resposta são lidos e enviados ao servidor de *background* para registrar *login* e

salvar dados em base Postgresql. 2. Menu inicial da aplicação é aberto com listagem de rotas.	
Nome do fluxo alternativo (extensão)	Descrição
Linha 4: retorno da resposta	4.1 Caso o usuário cancele a tentativa de *login*, uma falha de autenticação é exibida e a tela inicial de *login* é exibida.
Linha 1: pós-condição dados da resposta	1.1 Se ocorrer erro de autenticação, uma mensagem de alerta é exibida e a página inicial de *login* é reapresentada ao usuário.

Fonte: autoria própria.

No Quadro 17 é apresentado o caso de uso Listar rotas, que se subdivide em listar rotas próprias e listar rotas de terceiros.

Quadro 17 – Caso de uso listar rotas

Caso de uso: Listar rotas. **Descrição:** Esse caso de uso dispara um evento de consulta na base de dados de todas as rotas próprias ou de terceiros, de acordo com o escolhido pelo usuário. O retorno dessa consulta é uma listagem de rotas. **Evento Iniciador:** Clicar no botão listar rotas próprias ou listar rotas de terceiros. **Atores:** Usuário **Pré-condição:** Estar autenticado no sistema. **Sequência de Eventos:** 1. Ator clica no botão de Listar rotas. 2. Enviada requisição para o Serviço de consulta. 4. Serviço retorna resposta com dados de rotas. **Pós-Condição:** 1. Dados da resposta são lidos e apresentados em forma de lista para o usuário.	
Nome do fluxo alternativo (extensão)	Descrição
Linha 4: retorno da resposta	4.1 Caso ocorra algum erro resposta será vazia e navegador apresentara mensagem alertando a falha do carregamento da lista.

Fonte: autoria própria.

O Quadro 18 apresenta o caso de uso visualizar rotas, que apresenta ao usuário o percurso percorrido e também possível imagens capturadas durante o trajeto.

Quadro 18 – Visualizar rotas

Caso de uso: Visualizar rotas. **Descrição:** Exibe a rota percorrida em um mapa com o traçado do caminho e também com as imagens capturadas durante o percurso (caso existam). **Evento Iniciador:** Clicar no botão visualizar rotas na lista de rotas. **Atores:** Usuário **Pré-condição:** Existir Rota em uma das listas e usuário clicar no botão visualizar. **Sequência de Eventos:** 1. Ator clica no botão de visualizar rotas. 2. Enviada requisição de consulta de pontos e comentários de uma rota para o servidor. 4. Serviço retorna resposta com dados da rota consultada. 5. Página exibe um mapa com dados da rota. **Pós-Condição:** 1. Dados da resposta são lidos e apresentados em forma de lista para o usuário.

Nome do fluxo alternativo (extensão)	Descrição
Linha 4: retorno da resposta	4.1 Caso nenhuma informação exista para a rota consultada, mensagem informando sobre dados não encontrados será exibida no lugar do mapa.

Fonte: autoria própria.

No Quadro 19 é apresentada a expansão do caso de uso baixar rota, o qual possibilita um usuário a fazer *download* de uma rota para seu dispositivo móvel, para percorrê-la posteriormente ou para poder visualizar a rota em estado *off-line*.

Quadro 19 – Baixar rotas

Caso de uso: Baixar rota. **Descrição:** Efetua o *download* de uma rota específica para seu dispositivo móvel. **Evento Iniciador:** Clicar no botão baixar rota na lista de rotas. **Atores:** Usuário **Pré-condição:** Usuário clicar no botão baixar. **Sequência de Eventos:** 1. Ator clica no botão de baixar rota. 2. Comunicação com servidor é iniciada, persistindo a informação de *download* da rota em tabela apropriada. 4. Serviço retorna resposta com confirmação de sucesso. 5. Página exibe mensagem de sucesso. **Pós-Condição:** Lista de rota é exibida

Nome do fluxo alternativo (extensão)	Descrição

Linha 4: retorno da resposta	4.1 Caso rota não exista ou tenha ocorrido algum erro na persistência da requisição de *download*, mensagem informando sobre problema será exibida ao usuário.

Fonte: autoria própria.

No Quadro 20 é apresentado o caso de uso excluir rota, que faz a exclusão de uma rota do servidor *web*, porém ela continuará a existir nos dispositivos móveis dos usuários que a baixaram. A opção de exclusão só está disponível na listagem de rotas próprias, ou seja, o usuário somente poderá excluir rotas de sua autoria.

Quadro 20 – Excluir rotas

Caso de uso:
Excluir rota.
Descrição:
Efetua a exclusão de uma rota, e seus comentários, do sistema *web*.
Evento Iniciador:
Clique no botão excluir rota na lista de rotas.
Atores:
Usuário
Pré-condição:
Existir rotas na lista de rotas próprias.
Sequência de Eventos:
1. Ator clica no botão de excluir rota.
2. Ator confirma exclusão da rota.
2. Comunicação com servidor é iniciada, enviado requisição de exclusão da rota.
4. Serviço retorna resposta com confirmação de sucesso.
5. Página exibe mensagem de sucesso.
Pós-Condição:
Lista de rotas é exibida

Nome do fluxo alternativo (extensão)	Descrição
Linha 2: confirmação de exclusão	2.1 Caso o usuário não confirme a exclusão, a lista de rotas e reexibida.

Fonte: autoria própria.

No Quadro 21 está o caso de uso inserir comentário. Esse caso de uso faz a inclusão de um comentário em uma determinada rota. A opção de comentários fica disponível logo após o mapa de visualização da rota.

Quadro 21 – Incluir comentários

Caso de uso:
Incluir comentário.
Descrição:
Efetua a inclusão de um comentário em uma rota específica que está sendo visualizado.
Evento Iniciador:

Visualizar uma rota. **Atores:** Usuário **Pré-condição:** Existir rota. **Sequência de Eventos:** 1. Ator escreve comentário para a rota. 2. Ator clica no botão incluir comentário. 3. Comunicação com servidor é iniciada, enviado requisição de inclusão de comentário na rota. 4. Serviço retorna resposta com confirmação de sucesso. 5. Página exibe comentário inserido. **Pós-Condição:** Não há.

Nome do fluxo alternativo (extensão)	Descrição
Linha 2: clicar no botão incluir comentário.	2.1 Caso o comentário seja inválido, mensagem de erro é exibida e inclusão do comentário e cancelada.

Fonte: autoria própria.

No Quadro 22 é apresentado o caso de uso excluir comentário utilizado para excluir um comentário em uma determinada rota. A opção de comentários fica disponível após o mapa de visualização da rota.

Quadro 22 – Excluir comentário

Caso de uso: Excluir comentário. **Descrição:** Efetua a exclusão de um comentário em uma rota específica que se está sendo visualizado. A opção de exclusão de comentário somente está disponível para comentários inseridos pelo próprio usuário. **Evento Iniciador:** Visualizar rota. **Atores:** Usuário **Pré-condição:** Existir rota com comentário do usuário. **Sequência de Eventos:** 1. Ator clica no botão excluir comentário presente ao lado do seu comentário. 2. Ator confirma exclusão de comentário. 2. Comunicação com servidor é iniciada, enviado requisição de exclusão de comentário na rota. 4. Serviço retorna resposta com confirmação de sucesso. 5. Página exibe lista de comentários atualizada. **Pós-Condição:** Não há.

Nome do fluxo alternativo (extensão)	Descrição

Linha 2: confirmação de exclusão de comentário.	2.1 Caso o usuário não confirme a exclusão, nenhuma outra ação é executada.

Fonte: autoria própria.

No Quadro 23 está apresentado o caso de uso de *logout* do sistema, por meio dele o usuário se desvincula do sistema para finalizar o seu uso.

Quadro 23 – *Logout* do sistema

Caso de uso: Efetuar *logout*. **Descrição:** Efetua *logout* do usuário do sistema. **Evento Iniciador:** Clicar no botão de *logout* no cabeçalho da página. **Atores:** Usuário **Pré-condição:** Usuário estar autenticado no sistema. **Sequência de Eventos:** 1. Ator clica no botão efetuar *logout*. 2. Comunicação com servidor de *logout* do sistema é efetuada. 4. Serviço retorna resposta com confirmação de sucesso. 5. Página exibe tela de *login*. **Pós-Condição:** Não há.	
Nome do fluxo alternativo (extensão)	Descrição

Fonte: autoria própria.

3.2.3 Modelagem *Web Services*

O Quadro 24 apresenta a listagem dos requisitos funcionais identificados para o sistema *web service* responsável pela integração entre a aplicação *web* e o aplicativo *mobile*. O *web service* funciona apenas com três 'portas' de entrada e saída de requisição, uma para as requisições *web* e outra para as requisições do aplicativo móvel. Uma das 'portas' é responsável por registrar a autenticação do usuário na aplicação *web*, as outras duas são responsáveis, uma para atender as requisições de aplicação *web* e a outra para as requisições do aplicativo *mobile*. Cada uma dessas duas 'porta' recebe, por parâmetro, um arquivo JSON contendo a autenticação da requisição, um parâmetro que define qual método será executado e os dados de entrada para o método. A resposta também é um JSON, porém, com a resposta a requisição do cliente.

Quadro 24 – Requisitos funcionais *web service*

Identificação	Nome	Descrição
RF01	Método recebe requisição (recebeRequisicao)	É o método público de acesso ao *web service* pela aplicação *mobile*. Este método recebe um JSON do tipo Resposta contendo dados e, a partir desses dados, identifica qual método privado deverá executar. Após a execução a resposta devolvida ao cliente será um novo JSON também do tipo Resposta.
RF02	Método trata solicitação (trataSolicitacao)	É o método público de acesso ao *web service* pela aplicação *web*. Este método recebe um JSON do tipo Resposta contendo os dados e, a partir desses dados, identifica qual método privado deverá executar. Após a execução, a resposta devolvida ao cliente será um novo JSON também do tipo Resposta.

Fonte: autoria própria.

No Quadro 25 estão apresentados os requisitos não funcionais da aplicação *web service*, estes representam as funcionalidades privadas da aplicação que são orquestradas pelos requisitos funcionais.

Quadro 25 – Requisitos não funcionais *web service*

Identificação	Nome	Descrição
RNF01	Método para recuperar pontos (recuperaPontos)	É o método responsável por coletar e retornar todos os pontos, imagens e comentários pertencentes a uma rota específica recebida na requisição.
RNF02	Método listar (listar)	Método responsável por retornar todas as rotas obedecendo a um filtro recebido por parâmetro, o qual indica se serão retornadas rotas próprias ou rotas de terceiros ou todas as rotas com seus dados completos, ou ainda todas as rotas sem os dados de comentários, pontos ou imagens.
RNF03	Método para excluir rota (deletaRota)	Método responsável por excluir da base uma rota específica recebida via parâmetro e todos os seus dados vinculados.
RNF04	Método para agendar *download* de rota (AgendaDownloadRota)	Método responsável por coletar dados de todas as rotas que o usuário deseja baixar para seu aplicativo, empacotá-los em listas e retornar em um Json de Resposta.
RNF05	Método para incluir comentário (incComentario)	Método responsável pela inclusão de comentários em uma rota específica.
RNF06	Método para excluir comentário (excluirComentario)	Método responsável pela exclusão de um comentário específico de uma rota.
RNF07	Método para *download* de rotas (downloadRotas)	Método responsável por coletar dados de todas as rotas que o usuário deseja baixar para seu aplicativo, empacotá-los em listas e retornar em um Json de Resposta.
RNF08	Método para armazenar rotas (gravaRotas	Método que recebe uma ou mais rotas com seus dados de pontos e imagens e persiste na base de dados do *web*

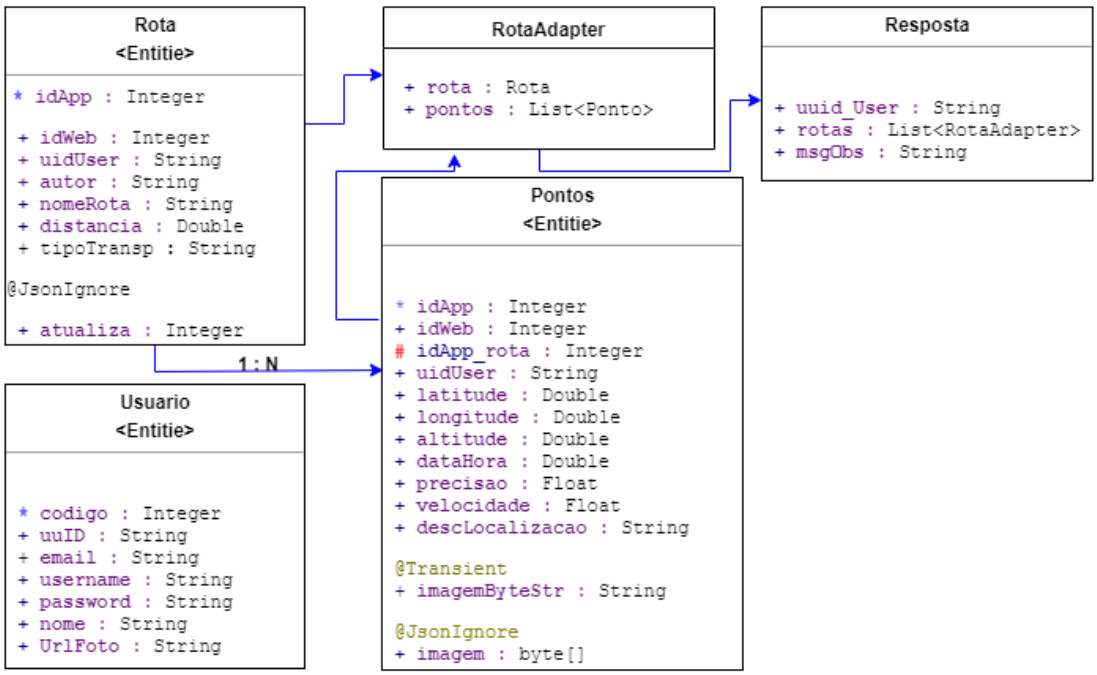
		service. Após a persistência, o método devolve ao cliente todas as rotas que foram gravadas com sucesso e seus respectivos "ids" da base de dados *web*, dessa forma consegue-se fazer o controle e garantir que uma mesma rota não seja gravada com mais de um "id" (gravação duplicada).
RNF09	Método para validar usuário (validaUsuario)	Método executado logo após o recebimento da requisição. No Json recebido há um campo que deve conter o Id do usuário que esta efetuando a requisição. Com essa informação é realizada uma consulta na base de dados *web* tentando identificar se tal usuário possui cadastro, caso possua o tratamento da requisição, caso contrário, é retornado uma resposta contendo um código de erro de autenticação.

Fonte: autoria própria.

As Figuras 3 e 4 apresentam os diagramas de entidades que representam as classes do banco de dados da aplicação e classes auxiliares de comunicação. A apresentação das classes foi dividida em quatro imagens distintas isolando cada parte do sistema, sendo eles: aplicativo *mobile*, *web service* de comunicação com aplicativo *mobile*, página *web* e *web service* de comunicação com o aplicativo *web*. A aplicação de *web service* foi dividida nos seus dois segmentos, *web* e *app*, para uma melhor visualização das classes utilizadas em cada serviço. As classes anotadas com "<Entitie>" representam classes de persistência e nessas classes campos anotados com "@Transient" representam campos auxiliares que não são persistidos.

A Figura 4 apresenta o diagrama de classes do aplicativo *mobile*, dessas classes apresentadas, Rota, Pontos e Usuário estão anotadas com "<Entitie>" indicando que essas representam modelo de entidades de persistência de dados na base de dados SQLite do Android. Os seus atributos que são marcados com * representam chave primária, com # representam chave estrangeira e o campo "@Transient" representa campos auxiliares que não são persistidos na base de dados. Já as demais classes são utilizadas como auxiliares na aplicação e também como modelo para o JSON de comunicação com o *web service*.

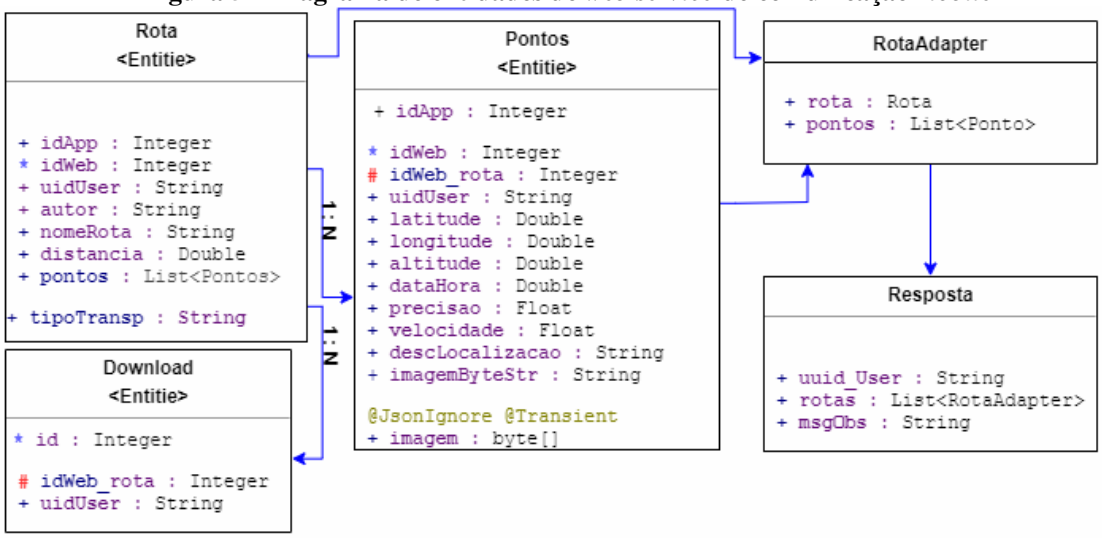
Figura 4 – Diagrama de entidades do aplicativo *mobile*



Fonte: autoria própria.

Na Figura 5 é apresentado o diagrama de entidades da parte do *web service* responsável pela comunicação com o aplicativo *mobile*. As classes anotadas também com "<Entitie>" representam que elas são classes modelo de persistência de dados na base de dados escolhida para a aplicação, neste caso, Postgresql. As demais classes são utilizadas também como auxiliares na aplicação e no Json de comunicação.

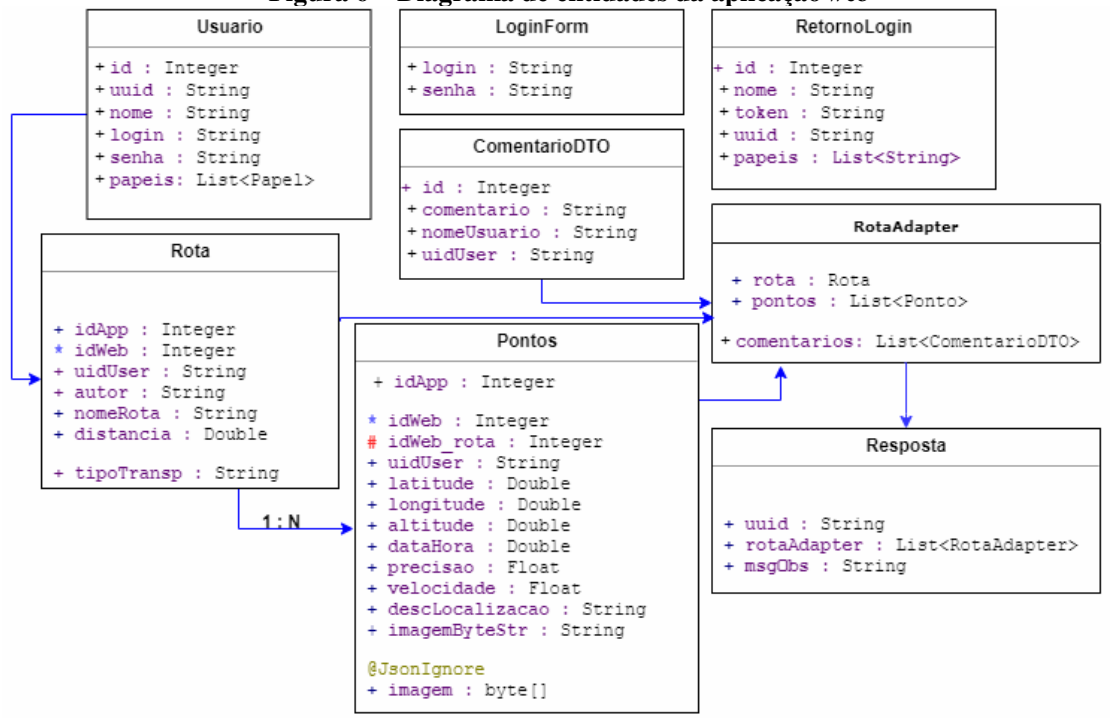
Figura 5 – Diagrama de entidades do *web service* de comunicação *mobile*



Fonte: autoria própria.

Na Figura 6 está o diagrama de classes da aplicação *web*. Nesse diagrama não são apresentadas as classes entidades de persistência, já que a aplicação *web* busca e envia seus dados em um *web service* e é este que trata a persistência de dados, ele é apresentado na Figura 7.

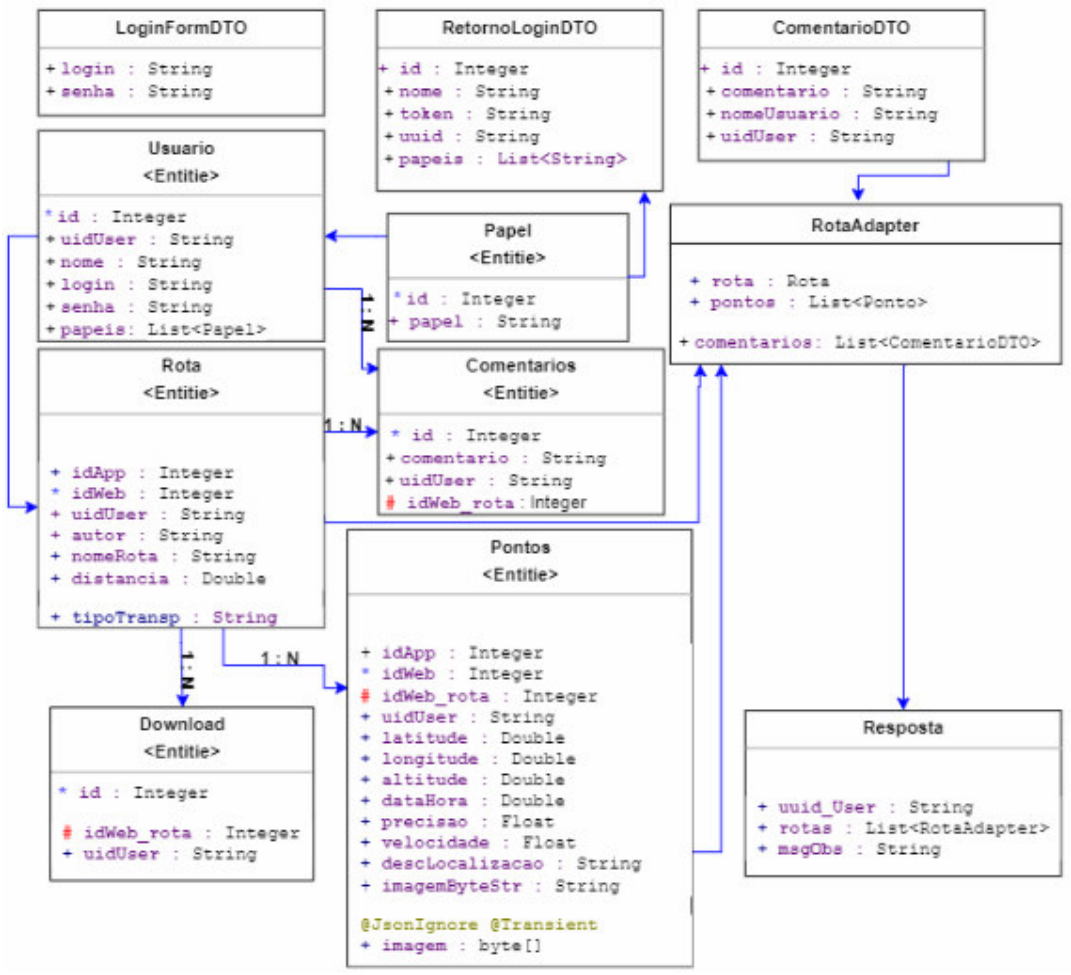
Figura 6 – Diagrama de entidades da aplicação *web*



Fonte: autoria própria.

Na Figura 7, é apresentado o diagrama de classes da parte do *web service* da aplicação *web*, com as classes de persistência e classes auxiliares utilizadas na comunicação.

Figura 7 – Diagrama de entidades do *web service* da aplicação *web*



Fonte: autoria própria.

No Quadro 26 estão os campos da entidade que representa a tabela de rota da aplicação. Uma rota possuirá pontos relacionados a ela (Quadro 27) que serão utilizados para sua exibição no mapa, bem como um nome, autor e uma distância total do ponto inicial ao final.

Quadro 26 – Campos da tabela rota

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
idApp	Numérico	Não	Sim	Não
idWeb	Numérico	Sim	Não	Não
uidUser	Texto	Não	Não	Não
Autor	Texto	Não	Não	Não
nomeRota	Texto	Não	Não	Não
distancia	Texto	Sim	Não	Não

tipoTransp	String	Não	Não	Não
Atualiza	Inteiro	Não	Não	Não

Fonte: autoria própria.

O Quadro 27 apresenta os campos da tabela de pontos que estão vinculados a uma rota (quadro 26).

Quadro 27 – Campos da tabela pontos

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
idApp	Numérico	Não	Sim	Não
idWeb	Numérico	Sim	Não	Não
IdApp_rota	Numérico	Não	Não	Sim
IdWeb_rota	Numérico	Não	Não	Sim
uidUser	Texto	Não	Não	Não
Latitude	Numérico	Não	Não	Não
longitude	Numérico	Não	Não	Não
Altitude	Numérico	Não	Não	Não
Precisão	Numérico	Não	Não	Não
velocidade	Numérico	Não	Não	Não
dataHora	Texto	Não	Não	Não
descLocalização	Texto	Sim	Não	Não
Imagem	Bite	Sim	Não	Não

Fonte: autoria própria.

O Quadro 28 apresenta a tabela de Usuário que é utilizada para armazenar dados dos usuários que já utilizaram o aplicativo no dispositivo *mobile* instalado.

Quadro 28 – Campos da tabela usuário

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
Código	Numérico	Não	Sim	Não
UuID	Texto	Sim	Não	Não
Email	Texto	Não	Não	Sim
Username	Texto	Não	Não	Não
Password	Texto	Não	Não	Não
Nome	Texto	Não	Não	Não
urlFoto	Texto	Não	Não	Não

Fonte: autoria própria.

No Quadro 29 está apresentada a tabela Usuário da aplicação *web*. Essa tabela possui diferença da tabela da aplicação *mobile*, pois na aplicação *web* há um controle de permissões de acesso às páginas, o que não ocorre na aplicação *web* e também porque não há integração entre essas tabelas.

Quadro 29 – Campos da tabela usuário da aplicação *web*

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
Id	Numérico	Não	Sim	Não
uidUser	Texto	Sim	Não	Não
Login	Texto	Não	Não	Não
Senha	Texto	Não	Não	Não
Nome	Texto	Não	Não	Não
Papeis	Lista de Texto	Não	Não	Não

Fonte: autoria própria.

No Quadro 30 estão descritos os campos da tabela de comentários.

Quadro 30 – Campos da tabela comentário

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
Id	Numérico	Não	Sim	Não
Comentario	Texto	Não	Não	Não
uidUser	Texto	Não	Não	Não
idWeb_Rota	Numérico	Não	Não	Não

Fonte: autoria própria.

No Quadro 31 estão descritos os campos da tabela de *download*.

Quadro 31 – Campos da tabela *download*

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
Id	Numérico	Não	Sim	Não
uidUser	Texto	Não	Não	Não
idWeb_Rota	Numérico	Não	Não	Não

Fonte: autoria própria.

No Quadro 32 estão descritos os campos da tabela de papéis.

Quadro 32 – Campos da tabela papéis

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
Id	Numérico	Não	Sim	Não
Papel	Texto	Não	Não	Não

Fonte: autoria própria.

As tabelas apresentadas na Figura 4 anotadas com "Entitie" representam, também, as classes de persistência do aplicativo *mobile*. Os campos anotados com "Transiente" se referem a campos auxiliares que não são persistidos e por isso não estão presentes na base de dados. A seguir são apresentadas as classes que se referem às classes persistentes do aplicativo *mobile*. O Quadro 33 apresenta a classe rota.

Quadro 33 – Descrição da classe rota

Identificação:	Rota
Descrição:	Registros de rotas e distância total delas
Requisitos:	RF01
Atributos:	idApp (número): identificador único, auto incremento para vincular os registros de pontos da rota no aplicativo *mobile*. idWeb (numérico): identificador de que a rota já foi compartilhada, seu valor é o identificador único da rota no banco de dados da aplicação *web*. uidUser (texto): identificador de qual usuário gravou essa rota. Esse identificador é retornado pelo Google no momento da autenticação e é único para cada usuário. Autor (texto): nome do usuário autor da rota. nomeRota (texto): nome da rota. distancia (número): distância total percorrida da rota. tipoTransp(texto): Tipo de transporte utilizado para efetuar o trajeto.(a pé, moto, bicicleta, carro, etc) Atualiza (texto): campo utilizado para marcar se uma rota está com suas descrições de pontos e distâncias atualizadas.

Fonte: autoria própria.

No Quadro 34 está a apresentação da classe pontos que armazena os pontos vinculados a uma rota.

Quadro 34 – Descrição da classe pontos

Identificação:	Pontos
Descrição:	Pontos de referências das rotas percorridas, identificadas por um ID de rota e uma sequência cronológica de gravação auto incremento.

Requisitos:	RF01
Atributos:	idApp (número): identificador único, auto incremento para vincular os registros de pontos da rota no aplicativo *mobile*. idWeb (numérico): identificador de que a rota já foi compartilhada, seu valor é o identificador único da rota no banco de dados da aplicação *web*. uidUser (texto): identificador de qual usuário gravou essa rota. Esse identificador é retornado pelo Google no momento da autenticação e é único para cada usuário. idApp_rota : identificador de qual rota o ponto pertence. latitude (numérico): latitude da localização. longitude(numérico): longitude da localização. altitude(numérico): altitude da localização. precisão(numérico): valor de 1 a 10 representando uma escala de quão exato é o ponto capturado com a posição real do usuário. É definido pela intensidade do sinal GPS e da quantidade de satélites usados na triangularização do ponto. velocidade(numérico): velocidade de deslocamento no momento da captura. dataHora(texto): data e hora da captura do ponto. descLocalizacao(texto) descrição do ponto de localização. Imagem (byte): imagem obtida (fotografia) vinculada ao ponto.

Fonte: autoria própria.

No Quadro 35 está a apresentação da classe usuário. Esses dados são retornados pelo Google no momento da autenticação.

Quadro 35 – Descrição da classe usuário

Identificação:	Usuário
Descrição:	Dados dos usuários que já se conectaram a aplicação por meio do aplicativo deste dispositivo.
Requisitos:	RF01
Atributos:	codigo (número): identificador único, auto incremento. uuID (texto): identificador único do usuário nas bases de dados do Google. Email(texto): *email* do usuário. username(texto): identificador do usuário. Password(texto): senha criptografada do usuário. nome(texto): nome do usuário. urlFoto(texto): URL da foto de perfil do usuário no Google.

Fonte: autoria própria.

As tabelas apresentadas na Figura 5 e 7 anotadas com “Entitie” representam, também, as classes de persistência do *web service* que atende tanto a requisições da aplicação *mobile*,

quanto a requisições da aplicação *web*. Os campos anotados com "Transiente" se referem aos campos auxiliares que não são persistidos e por isso não estão presentes na base de dados. A seguir são apresentadas as classes que se referem às classes persistentes do *web service*.

O Quadro 36 apresenta a classe rota.

Quadro 36 – Descrição da classe rota

Identificação:	Rota
Descrição:	Registros de rotas compartilhadas pelos usuários.
Requisitos:	RF01
Atributos:	idWeb (número): identificador único, auto incremento para vincular os registros de pontos da rota na base de dados *web*. idApp (numérico): identificador da rota no aplicativo de origem, seu valor é o identificador único da rota no banco de dados da aplicação *mobile* que a gerou. uidUser (texto): identificador de qual usuário gravou essa rota. Esse identificador é retornado pelo Google no momento da autenticação e é único para cada usuário. Autor (texto): nome do usuário autor da rota. nomeRota (texto): nome da rota. distancia (numérico): distância total percorrida da rota. tipoTransp(texto): meio utilizado para percorrer a rota.

Fonte: autoria própria.

O Quadro 37 apresenta a classe pontos.

Quadro 37 – Descrição da classe pontos

Identificação:	Pontos
Descrição:	Registros dos pontos vinculados à rota.
Requisitos:	RF01
Atributos:	idWeb (número): identificador único, auto incremento para vincular os registros de pontos da rota a rota na base de dados *web*. idApp (numérico): identificador do ponto no aplicativo de origem, seu valor é o identificador único do registro no banco de dados da aplicação *mobile* que a gerou. uidUser (texto): identificador de qual usuário gravou a rota a qual o ponto pertence. Esse identificador é retornado pelo Google no momento da autenticação e é único para cada usuário. idWeb_Rota (numérico): identificador da rota a qual o ponto pertence. distancia (numérico): distância total percorrida da rota. latitude (numérico): latitude da localização. longitude(numérico): longitude da localização. altitude(numérico): altitude da localização. precisão(numérico): valor de 1 a 10 representando uma escala de quão exato é o ponto capturado com a posição real do usuário. É definido pela intensidade do sinal GPS e da quantidade de satélites usados na

	triangularização do ponto. velocidade(numérico): velocidade de deslocamento no momento da captura. dataHora(texto): data e hora da captura do ponto. descLocalizacao(texto) descrição do ponto de localização. imagemByteStr(texto): Imagem convertida em Base64.

Fonte: autoria própria.

O Quadro 38 apresenta a classe *download*.

Quadro 38 – Descrição da classe *download*

Identificação:	Download
Descrição:	Registros de solicitações de *download* de rotas realizadas pelos usuários.
Requisitos:	RF01
Atributos:	id (número): identificador único, auto incremento da solicitação de *download* na base de dados *web*. idWeb_Rota (inteiro): identificador da rota a qual a solicitação se refere. uidUser (texto): usuário que solicitou o *download*.

Fonte: autoria própria.

O Quadro 39 apresenta a classe usuário.

Quadro 39 – Descrição da classe usuário

Identificação:	Usuário
Descrição:	Dados dos usuários que já se conectaram a aplicação *web*.
Requisitos:	RF01
Atributos:	Id (número): identificador único, auto incremento. uidUser (texto): identificador único do usuário nas bases de dados do Google. login(texto): identificador do usuário. senha(texto): senha criptografada do usuário. nome(Texto): nome do usuário. papeis(lista de textos): identifica as permissões de acesso do usuário dentro da aplicação *web*.

Fonte: autoria própria.

O Quadro 40 apresenta a classe papeis.

Quadro 40 – Descrição da classe papéis

Identificação:	Papeis
Descrição:	Dados identificadores das permissões possíveis dentro da aplicação

	web.
Requisitos:	RF01
Atributos:	Id (número): sequencial de registro, auto incremento. papel(texto): identificador da permissão do usuário (ADMIN,USER, ...).

Fonte: autoria própria.

O Quadro 41 apresenta a classe comentários.

Quadro 41 – Descrição da classe comentário

Identificação:	Comentários
Descrição:	Registros de comentários feitos pelos usuários que visualizaram a rota compartilhada.
Requisitos:	RF01
Atributos:	id (número): identificador único, auto incremento do comentário na base de dados *web*. Comentário (texto): comentário. idWeb_Rota (inteiro): identificador da rota a qual o comentário se refere. uidUser (texto): usuário que fez o comentário.

Fonte: autoria própria.

3.3 APRESENTAÇÃO DO SISTEMA

A seguir está a apresentação do sistema que ocorre pela descrição das suas funcionalidades principais, apresentadas por meio das suas telas. Inicialmente é apresentado o aplicativo *mobile* e em seguida a aplicação *web*.

3.3.1 Apresentação do aplicativo *mobille*

O leiaute do aplicativo *mobile* é composto por seis telas: a primeira, que é apresentada ao iniciar o aplicativo, é conhecida como *splash* e está na Figura 8. Essa é uma tela simples, somente com o logo do sistema e fica ativa apenas por um segundo.

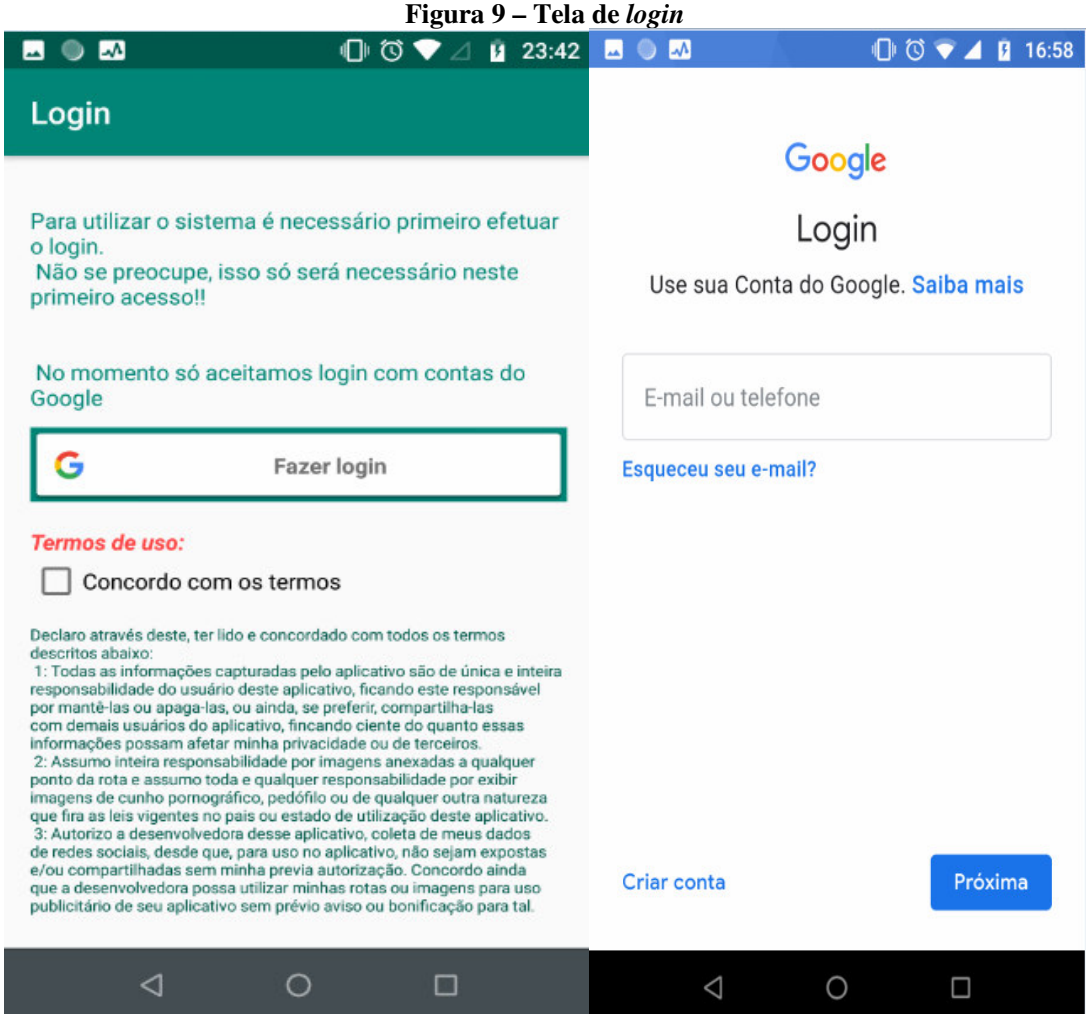
Figura 8 – Tela *splash*

Fonte: autoria própria.

Após um segundo, a tela *splash* é fechada e caso nenhum usuário esteja com a autenticação salva, a tela de termos e *login* é exibida (apresentada na Figura 9), caso contrário, a tela principal da aplicação é exibida, como mostra a Figura 10.

Na Figura 9 está a tela de autenticação do aplicativo. Essa tela possui um botão para *login* via API Google e uma lista contendo textos de termos de uso, além da opção para concordar com os termos de uso. Concordando com os termos e clicando em “Fazer Login”, é exibida a segunda tela da Figura 9, solicitando os dados de autenticação. Essa tela é gerada pela API Google de forma padrão e automatizada.

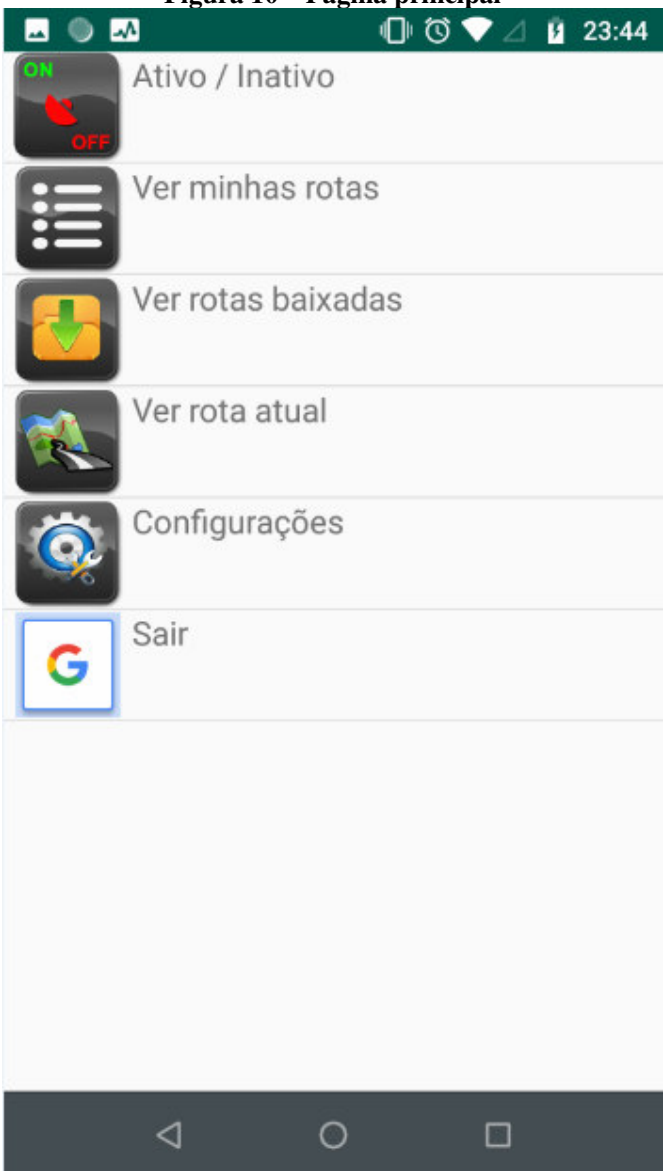
Figura 9 – Tela de *login*



Fonte: autoria própria.

Após o *login* efetuado é exibida a tela de menu principal da aplicação, apresentada na Figura 10. Nessa tela é apresentado um menu de opções básicas para utilização da aplicação. Ainda durante a apresentação dessa tela, são iniciados dois serviços. Um deles é o serviço que atualizará o cálculo das distâncias das rotas gravadas no banco, caso existam. Também neste serviço são atualizadas as descrições dos pontos de rotas salvos que ainda não possuem a descrição da localização. Essa descrição é obtida por meio de uma conexão com o servidor da Google e necessita de conexão com a Internet. O segundo serviço iniciado é o responsável por verificar, em intervalos determinados de tempo, se há rotas para serem baixadas da aplicação *web*.

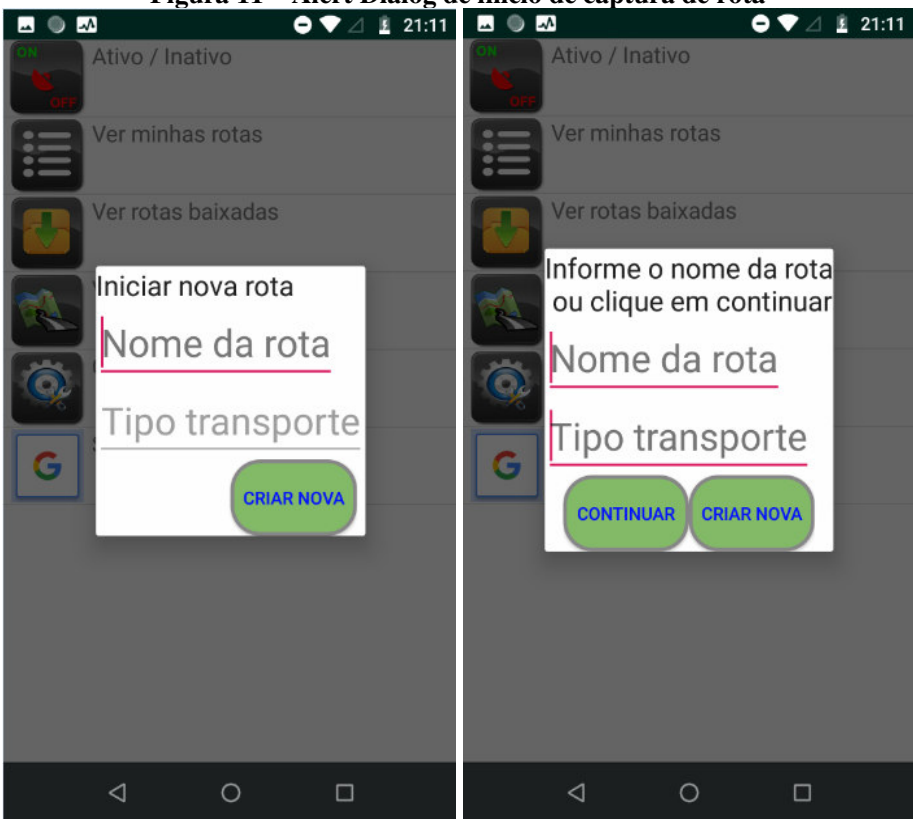
Figura 10 – Página principal



Fonte: autoria própria.

A tela do menu principal é composta por seis opções. A primeira delas é a responsável por iniciar um rastreamento (uma rota) ou continuar uma rota já existente. Ao selecioná-la será exibido um componente alert-Dialog solicitando um nome para a nova rota e qual o meio de locomoção utilizado para percorrê-la. Também nessa tela está disponível um botão continuar para que o usuário possa continuar a última rota que ele estava percorrendo. Caso não existam rotas salvas, a opção continuar é desabilitada. O componente alert-Dialog é exibido na Figura 11.

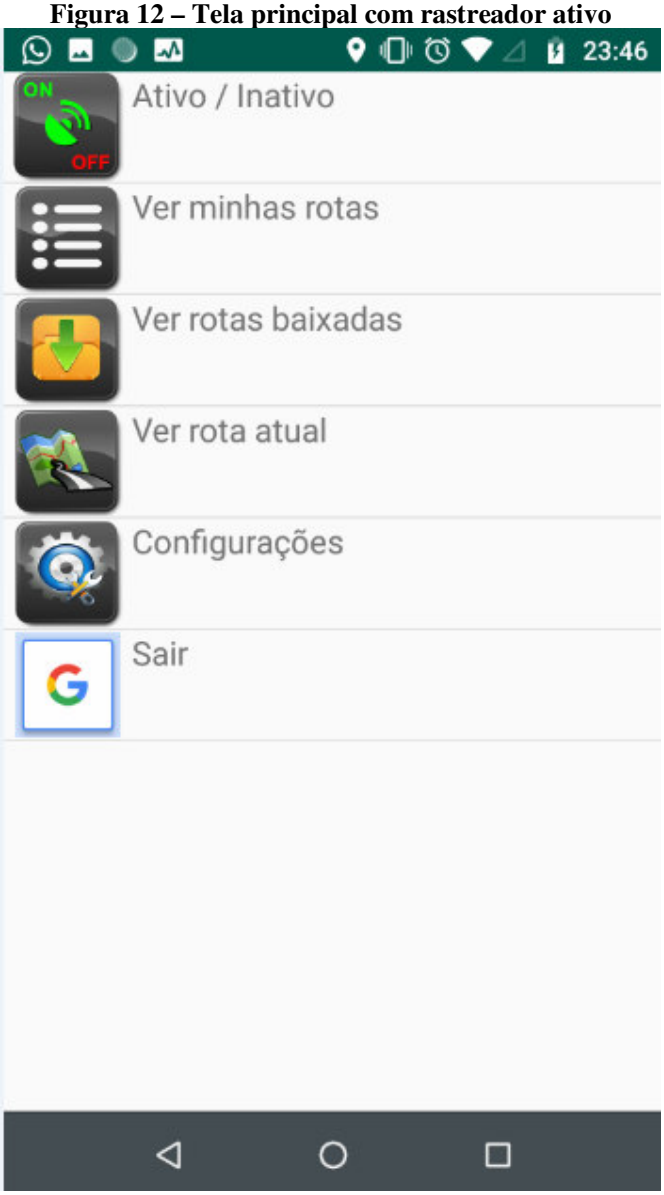
Figura 11 – Alert Dialog de início de captura de rota



Fonte: autoria própria.

Após selecionada a opção “Continuar” ou a opção “Criar Nova”, a tela principal é reapresentada com o ícone da opção Ativo/Inativo em cor verde, representando que o serviço de captura de rota está em funcionamento. Também é exibido na barra de *status* do dispositivo um ícone representando a atividade do sensor GPS. A Figura 12 apresenta como é exibida a tela principal quando o rastreamento se encontra ativo.

Figura 12 – Tela principal com rastreador ativo

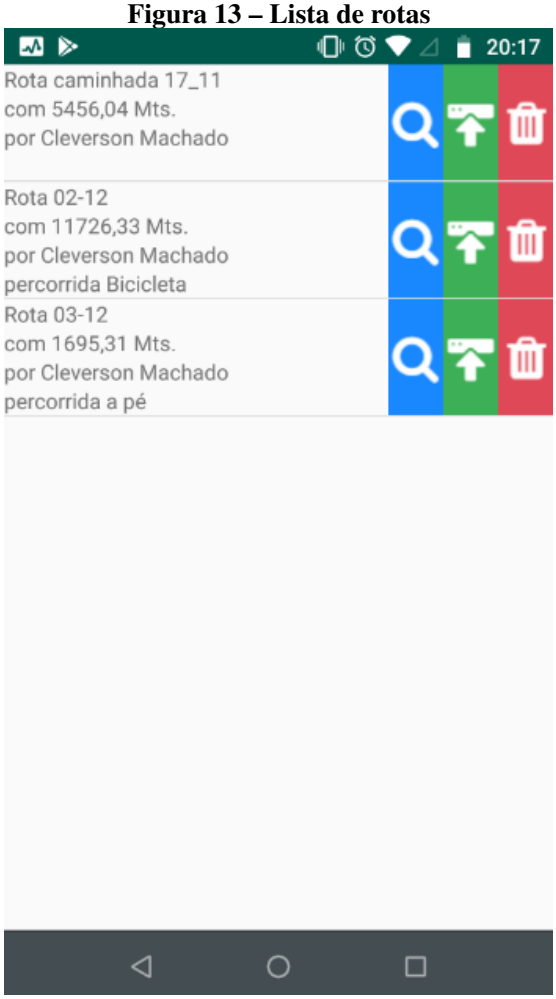


Fonte: autoria própria.

A segunda opção do menu principal exibe, por meio de uma lista ou ListView, todas as rotas, do usuário autenticado, salvas na base de dados do aplicativo *mobile* (como mostrado na Figura 13), bem como a informação da distância percorrida e o autor de cada rota.

A terceira opção (“Ver rotas baixadas”) é muito semelhante à segunda, porém exibe na lista somente as rotas de outros usuários que foram baixadas no aplicativo. Nessa opção de visualização da lista de rotas, a opção de compartilhamento fica desabilitada.

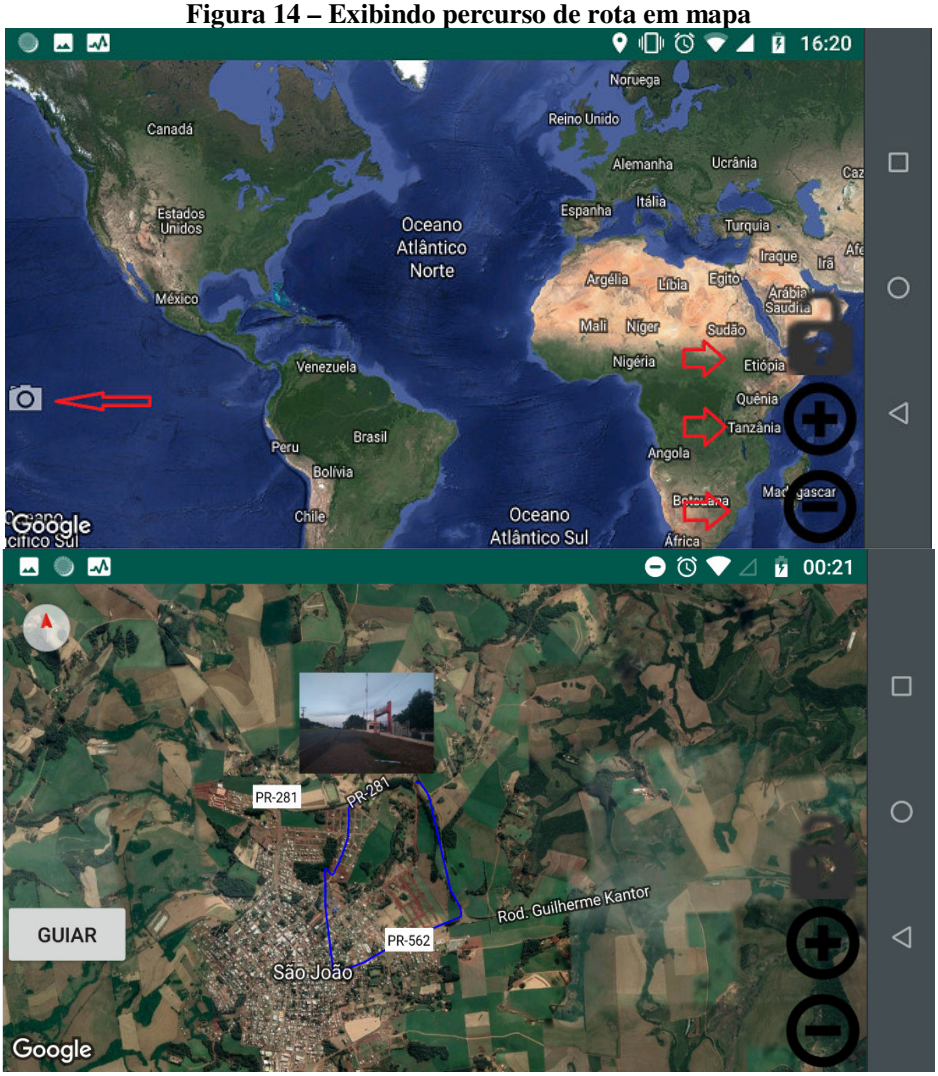
Figura 13 – Lista de rotas



Fonte: autoria própria.

Ao lado de cada rota são disponibilizadas três opções em forma de botão. O primeiro botão, com ícone de lupa significa “Ver” e, ao ser clicado, exibe os pontos da rota selecionada em um mapa (como mostra a Figura 14). O segundo, com ícone de u*pload* significa “Publicar” e, ao ser clicado, envia a rota selecionada, após confirmação, para o servidor *web*, e esta fica visível para todos os usuários da nuvem. O terceiro é a opção de exclusão que, após confirmação, exclui a rota do aplicativo do usuário. A exclusão no aplicativo não excluir rotas compartilhadas na nuvem, essas devem ser excluídas pelo gerenciador *web*.

Figura 14 – Exibindo percurso de rota em mapa



Fonte: autoria própria.

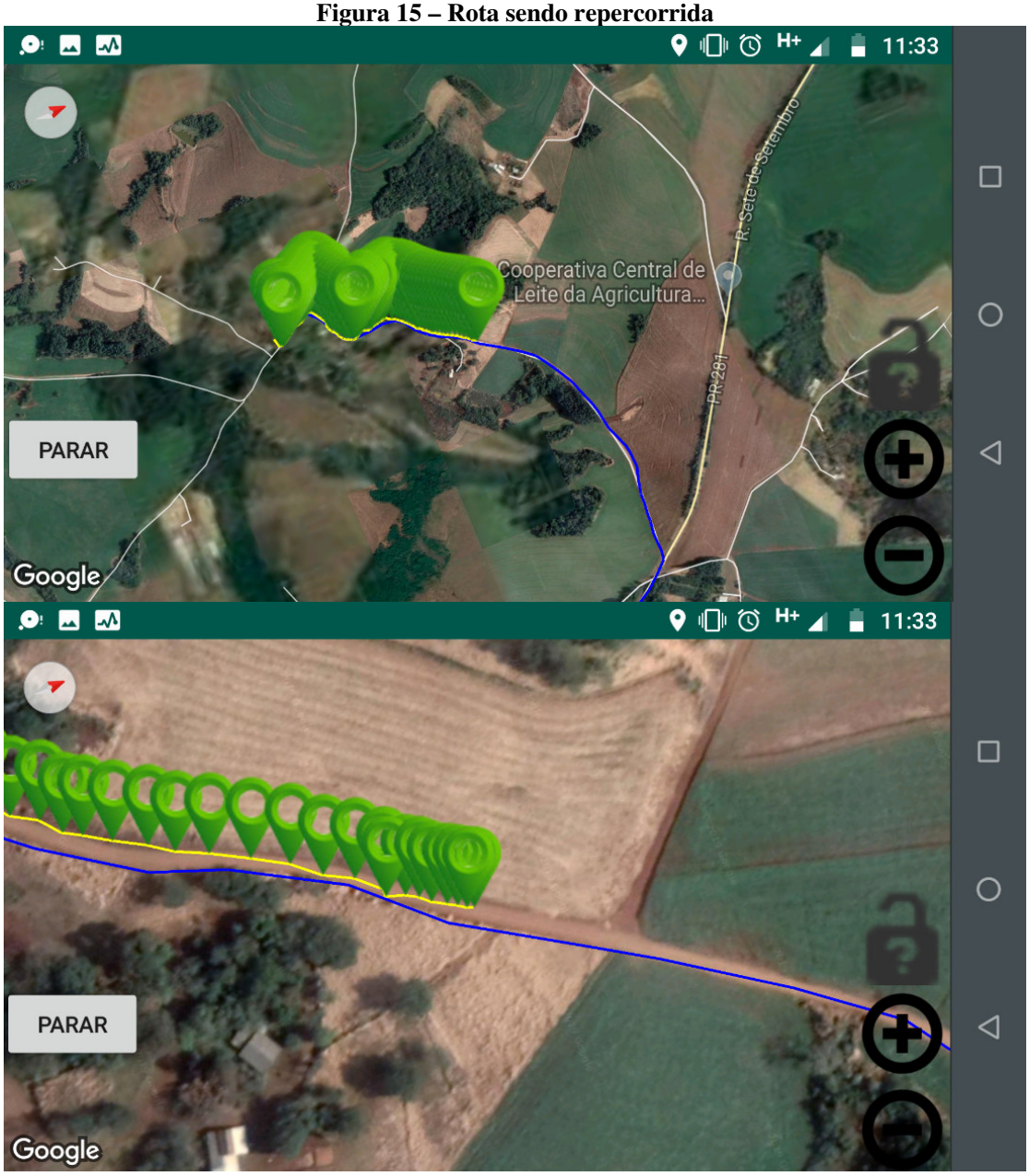
Na exibição do mapa, são inseridos os pontos da rota selecionada interligados com seu antecessor e sucessor por uma linha. Como mostrado por setas na Figura 14, o mapa também possui dois botões de *zoom*, (um para aumentar e outro para diminuir o nível de *zoom* de visualização), um botão de bloqueio para travar o *touch screen* e, caso a rota visualizada esteja sendo percorrida no momento, é exibido um botão para captura de imagem, caso contrário, é exibido um botão com a descrição ‘Guiar’ Esse botão fica visível somente quando nenhuma rota está sendo capturada.

A função do botão “Guiar” é possibilitar que o usuário possa refazer a rota que está sendo visualizada. Ao ser clicado, sua descrição muda para “Parar” e um novo processo interno é iniciado. Esse processo executa em paralelo e captura o ponto que o usuário se

encontra.

Após a captura, esse ponto é inserido no mapa como um marcador informando a posição atual do usuário. Esse processo de captura dos pontos e adição no mapa fica em execução até que o usuário saia da aplicação ou selecione a opção “Parar”. A Figura 15 apresenta uma rota sendo repercorrida, o traçado em azul representa a rota guia e o traçado em amarelo com marcadores verdes representa a posição em que o usuário está e por onde ele já passou. Nessa tela podem ser visualizados os marcadores diferenciados.

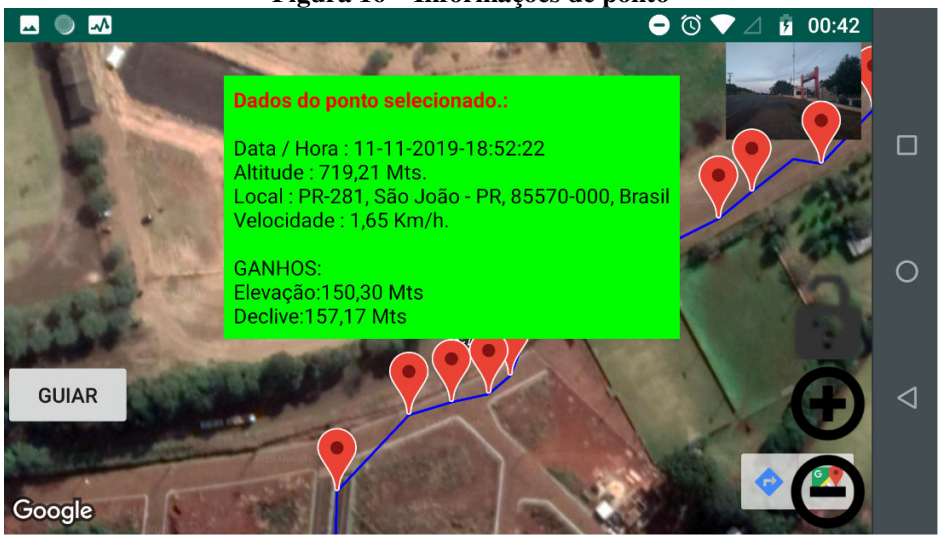
Figura 15 – Rota sendo repercorrida



Fonte: autoria própria.

Ao ser visualizada uma rota, selecionando-se um marcador, serão exibidas informações como, data e hora da captura da posição, descrição do endereço, velocidade aproximada de deslocamento, ganhos de aclive e declive, até o ponto selecionado. A Figura 16 apresenta essas informações em um ponto de uma rota. Ressalta-se que para que os pontos sejam exibidos no mapa, é necessário habilitar a apresentação deles na tela de configurações, como mostrado na Figura 17.

Figura 16 – Informações de ponto



Fonte: autoria própria.

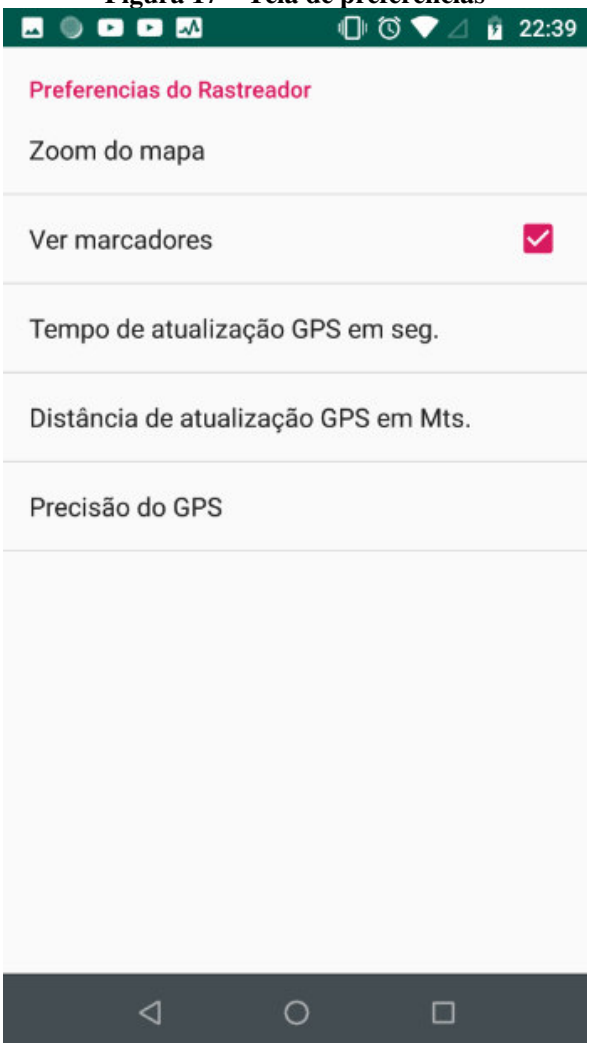
Voltando à tela principal existe, também, a opção “Ver rota atual” que exibe o mesmo mapa da Figura 14. Essa opção apresenta a última rota percorrida ou a rota que está sendo percorrida no momento, sendo que, neste caso, novos pontos que venham a ser capturados serão automaticamente inseridos ao mapa, possibilitando que o usuário acompanhe de forma gráfica o deslocamento.

Nesse modo não é possível utilizar a opção “Guiar”. Caso se esteja percorrendo a rota no momento, estará disponível um ícone fotográfico no canto inferir esquerdo da tela (mostrado na Figura 14), sendo possível capturar fotografias com a câmera e adicioná-las a rota em percurso.

Ainda nas opções da tela principal, está a opção de acesso às configurações do aplicativo. Quando selecionada, uma tela de preferências é exibida como mostra a Figura 17. Nessa tela são realizados ajustes, como distância de captura de posição entre dois pontos, tempo mínimo entre cada captura, precisão de captura, *zoom* de visualização padrão do mapa

e opção de exibição de marcadores de pontos no mapa.

Figura 17 – Tela de preferências



Fonte: autoria própria.

Finalizando as opções da tela principal, está a opção de o usuário sair do aplicativo para que outro usuário possa autenticar-se. Essa opção efetua o *logoff* da conta do Google e finaliza todos os serviços em execução da aplicação.

3.3.2 Apresentação da aplicação *web*

O leiaute da aplicação *web* é composto por quatro telas: a primeira, que é apresentada ao iniciar o aplicativo, é a tela de solicitação de autenticação. Essa tela é composta apenas de um botão que, ao ser clicado, consome a API Google de autenticação que solicitará os dados

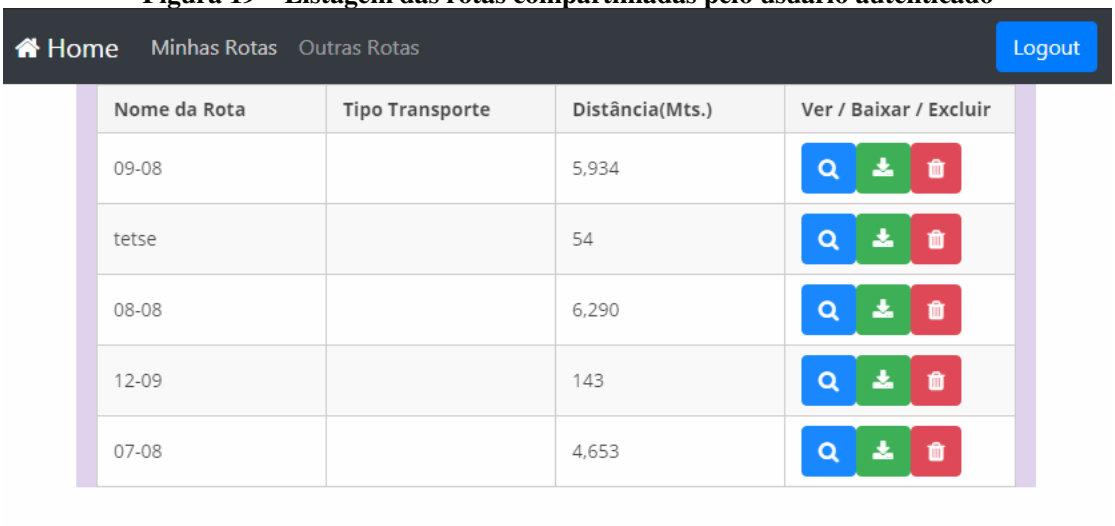
de *login* e senha. A Figura 18 apresenta a tela inicial da aplicação *web*.

Figura 18 – Tela inicial aplicação *web*

Fonte: autoria própria.

Após o *login* efetuado com sucesso, será exibida a tela principal da aplicação *web* qu exibe a listagem das rotas compartilhadas pelo usuário autenticado, como mostra a Figura 19. Nessa tela estão apresentadas todas as rotas compartilhadas pelo usuário que está autenticado, a partir dessa tela, é possível visualizar, baixar ou excluir qualquer uma das suas rotas. Essa tela também está acessível no menu superior, na opção “Minhas Rotas”.

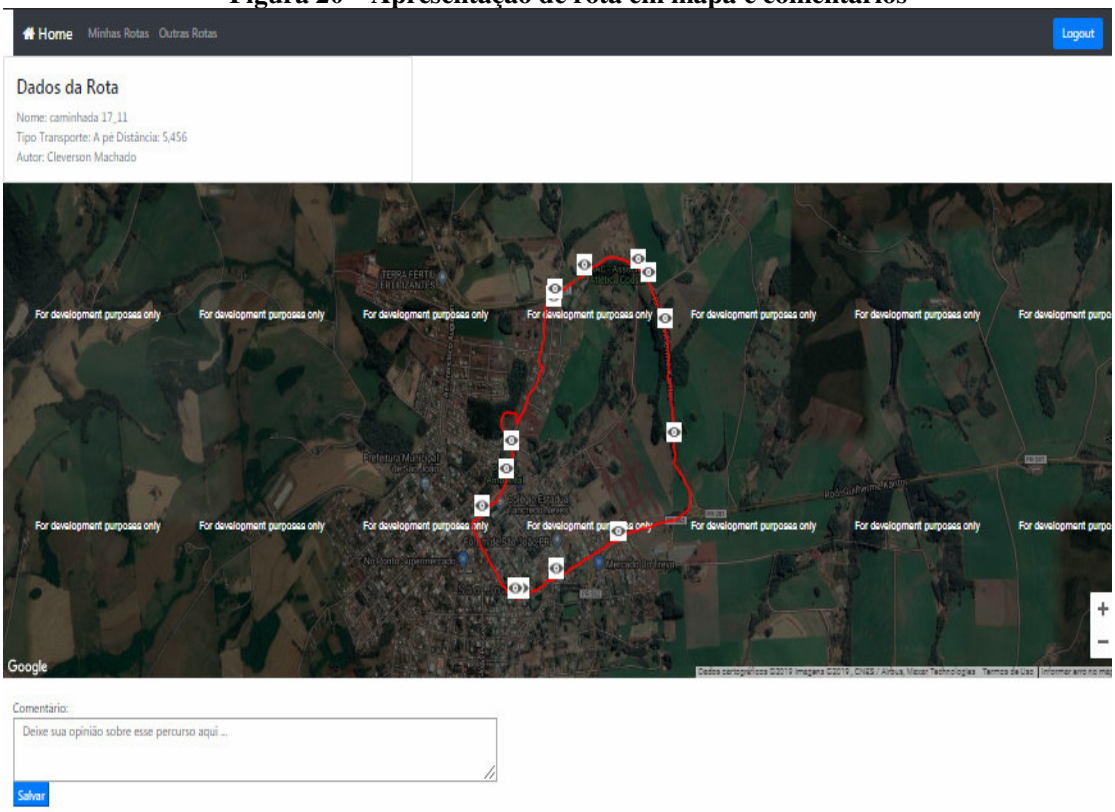
Figura 19 – Listagem das rotas compartilhadas pelo usuário autenticado



Fonte: autoria própria.

Na coluna mais a direita há três botões que significam, da esquerda para a direita respectivamente, visualizar, baixar e excluir. Clicando em visualizar é apresentada a rota percorrida em um mapa como apresentado na Figura 20.

Figura 20 – Apresentação de rota em mapa e comentários



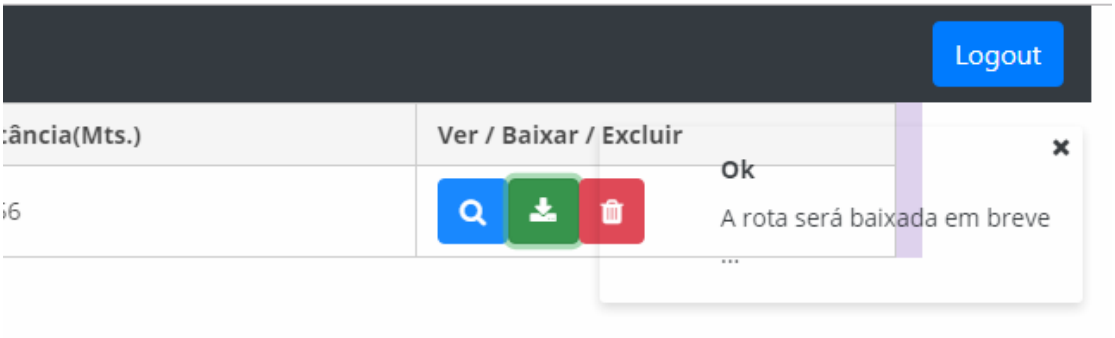
Fonte: autoria própria.

Pelas novas políticas de uso da API de mapas da Google, é exibido sobre o mapa uma tarja informando sobre o uso gratuito somente em ambiente de desenvolvimento de aplicações, mas para uso em produção é necessária a ativação de faturamento por licença de acesso. Em produção, a API é paga.

Na Figura 20 é apresentada uma rota compartilhada, na parte inferior da página há a opção para que os usuários compartilhem opiniões sobre o trajeto. O usuário pode visualizar os comentários de outros usuários e incluir comentários, mas excluir somente os comentários por ele incluídos.

Voltando às opções da listagem de rotas compartilhadas (apresentado na Figura 19), há a opção de baixar uma rota para seu dispositivo *mobile*, basta clicar sobre o ícone corespondente, que o sistema fará o agendamento do *download* da rota para o aplicativo e exibirá uma mensagem de confirmação ou falha. A Figura 21 apresenta essa mensagem.

Figura 21 – Confirmação de *download*



Fonte: autoria própria.

Ainda nas opções da listagem, está disponível a opção para que o usuário possa excluir uma de suas rotas. Essa ação pode ser feita por meio da listagem de suas rotas no ícone específico de exclusão.

Voltando ao *menu* superior da aplicação, existe a opção "Outras Rotas". Por meio dessa opção é possível visualizar as rotas compartilhadas pelos demais usuários e inserir comentários nas rotas. Na Figura 22 é apresentada essa listagem.

Figura 22 – Rotas de outros usuários

Fonte: autoria própria.

Ressalta-se que diferentemente das rotas próprias compartilhadas pelo usuário, nessa listagem não há a opção de exclusão de rota, já que elas não são de sua autoria. As opções de visualização, comentários e *download* de rotas de outros usuários funcionam da mesma forma que foi apresentado para as rotas próprias.

Finalizando as opções do *menu* superior, está a opção de *logout* da aplicação, ao clicar nessa opção, o usuário é desconectado da aplicação e redirecionado para a tela de *login* novamente.

3.4 IMPLEMENTAÇÃO DO SISTEMA

Para um melhor entendimento dessa sessão, o conteúdo foi subdividido em três etapas, sendo elas: implementação aplicativo *mobile*, implementação *web service* e implementação aplicação *web*.

3.4.1 Implementação do aplicativo *mobile*

Como apresentado na Seção 3.3, o aplicativo conta com seis telas, sendo uma tela inicial (*splash*), uma tela de *login* e uma tela principal com *links* de acesso para as demais telas operacionais do sistema.

Quando a tela inicial é apresentada, é validado se a última sessão do aplicativo foi encerrada com *logoff* do usuário, ou se o mesmo continua autenticado na aplicação *mobile* e, caso ele ainda esteja autenticado, o menu principal do aplicativo é aberto.

Caso o usuário não esteja autenticado, a aplicação abre a tela de *login* na qual o usuário deve concordar com os termos de uso e clicar na opção para fazer *login*. Na versão atual do aplicativo, somente está disponível o *login* por meio de conta Google e, para isso, o sistema utiliza a API Google Auth fornecida pela Google. Essa API se responsabiliza em solicitar as informações de usuário e senha, realizar a autenticação e devolver a aplicação de origem o *status* que pode ser falha ou sucesso na autenticação. Caso a resposta seja sucesso, também é retornado um objeto com informações do usuário autenticado, como *email*, nome e userId.

Para utilização da API Google Auth na aplicação é necessário que a classe da *Activity* implemente, GoogleApiClient.OnConnectionFailedListener e View.OnCreateContextMenuListener além de instanciar objetos de duas classes, GoogleApiClient e GoogleSignInClient.

Na Listagem 1 está apresentada a classe de configuração da API de autenticação e na Listagem 2 a chamada da API de autenticação e recuperação do resultado do *login.* Ressalta-se que antes da chamada da *intent* há a validação do aceite dos termos de uso que, na Listagem 2 foi suprimida.

Listagem 1 – Configuração da API de autenticação

```
private void configureLoginGoogle() {
   GoogleSignInOptions gop = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                .requestEmail().build();
  googleApiClient = new GoogleApiClient.Builder(this)
                .enableAutoManage(this, this)
                .addApi(Auth.GOOGLE_SIGN_IN_API, gop)
                .build();

  googleSignInClient = GoogleSignIn.getClient(this, gop);
}
```

Fonte: autoria própria.

Listagem 2 – Chamada e resposta da API de autenticação mobile

```
public void btnGoogleSignInLogin() {
        Intent signIIntent =
Auth.GoogleSignInApi.getSignInIntent(googleApiClient);
        startActivityForResult(signIIntent, 9001);
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
@Nullable Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   if(requestCode == 9001){
     GoogleSignInResult gResult =
Auth.GoogleSignInApi.getSignInResultFromIntent(data);
  GoogleSignInAccount conta = null;
  if(gResult.isSuccess()) {
     conta = gResult.getSignInAccount();
  }

  processaRetornoLogin(conta);
  }
}
```

Fonte: autoria própria.

Para que tudo isso funcione também é preciso adicionar a dependência do gms:play-services-auth e o *plugin* gms:google-services ao *gradle*, também é necessário adicionar ao *manifest* a permissão de Internet.

O método processaRetornoLogin recebe via parâmetro o objeto retornado pela API de autenticação com os dados da autenticação do usuário, a validação da autenticação efetuada dentro desse método e caso a autenticação tenha ocorrido com sucesso, os dados do usuário são salvos na base local e a tela principal da aplicação é aberta, caso contrário, a tela de *login* é novamente apresentada juntamente com uma mensagem de erro informando de *login*.

Na Listagem 3 é exibido o trecho de código responsável por validar a autenticação e na Listagem 4 o método auxiliar utilizado por criar o objeto usuário e persisti-lo na base de dados. Os objetos UsuarioLogado e googleSignInClient vistos na Listagem 4 são declarados como privados e estáticos tendo métodos de acesso configurados garantindo acesso dentro de toda a aplicação aos dados do usuário.

Listagem 3 – Validação de login

```
if(conta != null){
  boolean sucesso = UserAutenticate.init(us,
googleSignInClient);

  if(sucesso) { Intent i = new Intent(this, MainActivity.class);
                startActivity(i);
                finish();
              }
}
```

Fonte: autoria própria.

Listagem 4 – Criando sessão e persistência de usuário

```
public static synchronized boolean init(Usuario us,
GoogleSignInClient googleSignInClientt) {
        usuarioLogado = us;
        googleSignInClient = googleSignInClientt;
        Usuario uss = null;
        BD db = new BD("WR"); // Write
        try {
            uss = db.incereUsuario(us);
        } catch (Exception e) {
            e.printStackTrace();
        }
        db.close();
        if(uss == null) {
            logout();
            return false;
        }
        else
           return true;
}
```

Fonte: autoria própria.

Ainda na tela de *login* está subscrito o método onStart. Nesse método, apresentado na Listagem 5, é feita a validação de sessão ativa e de usuário autenticado. Essa validação serve para que quando o usuário abrir o aplicativo, se ele não fez *logoff* anteriormente, a aplicação abra automaticamente sem solicitar um novo *login*.

Listagem 5 – Validação de sessão e usuário autenticado sem *login*

```
@Override
public void onStart() {
   super.onStart();
   GoogleSignInAccount conta =
GoogleSignIn.getLastSignedInAccount(this);
        processaRetornoLogin(conta);
}
```

Fonte: autoria própria.

Após autenticado, o usuário é direcionado para a página inicial da aplicação *mobile* (Figura 12).

A tela principal da aplicação é desenhada utilizando um leiaute do tipo ListView. O sistema possui modelos de lista pré-definidas, porém esses modelos são limitados em termos de recursos, por isso, a tela principal foi personalizada com a criação de um componente do tipo *adapter*, possibilitando maior flexibilidade na adição de componentes à tela de listagem. O componente *adapter* desenvolvido é apresentado na Listagem 6.

Listagem 6 – *Adapter* personalizado

```
public menuInicialAdapter(Context context, MatrixCursor cursor,
Boolean ativo) {  this.context = context;
    this.cursor  = cursor;
    this.ativo   = ativo;
    inflater     = (LayoutInflater)
context.getSystemService(context.LAYOUT_INFLATER_SERVICE);
}
@Override
public int getCount() {  return cursor.getCount();
@Override
public Object getItem(int position) {
cursor.moveToPosition(position);  return cursor; }
@Override
public long getItemId(int position) {
    cursor.moveToPosition(position);    return cursor.getInt(0); }
@Override
public View getView(final int position, View convertView, ViewGroup
parent) {
    View elemento_lista = inflater.inflate(R.layout.menu_inicial,
null);
    setView(elemento_lista);
    TextView tvOpcao = (TextView)
elemento_lista.findViewById(R.id.tvOpcao);
    TextView tvDescricao = (TextView)
elemento_lista.findViewById(R.id.tvDescricao);
    ImageView ivIcon = (ImageView)
elemento_lista.findViewById(R.id.ivIcon);
    cursor.moveToPosition(position);
    tvOpcao.setText(cursor.getString(1));
```

```
    tvDescricao.setText(cursor.getString(2));
    if(cursor.getDouble(0) == 1 && ativo)
ivIcon.setImageResource(R.drawable.on);
 else if (cursor.getDouble(0) == 2 )
ivIcon.setImageResource(R.drawable.lista_rotas);
 else if (cursor.getDouble(0) == 3 )
ivIcon.setImageResource(R.drawable.lista_rotas_baixadas);
 else if (cursor.getDouble(0) == 4 )
ivIcon.setImageResource(R.drawable.rota_atual);
 else if (cursor.getDouble(0) == 5 )
ivIcon.setImageResource(R.drawable.config);
 else if (cursor.getDouble(0) == 6 )
ivIcon.setImageResource(R.drawable.common_google_signin_btn_icon_dark_focus
ed);
 return elemento_lista;
```

Fonte: autoria própria.

A construção de um *adapter* é realizada com o uso de uma classe que estenda BaseAdapter, que obriga a implementação de quatro métodos além do método construtor. Os métodos são: getCount() que retorna a quantidade de itens da listagem; getItem() que pode retornar um índice do item selecionado ou nesse caso um objeto que representa o item selecionado; getItemId() que retorna um identificador do tipo índice que representa a posição do item selecionado na lista; e o método principal do *adapter*, getView(). É nesse método que o leiaute de cada item da lista é desenhado e cada componente é inserido na sua devida posição, com a ajuda de um *eXtensible Markup Language* (XML) de leiaute. Esse método é chamado na inserção de cada item da lista.

Ao visualizar a tela principal, o usuário tem várias opções. Ao selecionar uma delas, o sistema identifica, por meio de um método da lista de opções, qual opção foi selecionada e direciona para a tela correspondente.

Ao ser iniciada a tela inicial da aplicação, também são ativados dois serviços, um responsável pela atualização dos dados das rotas salvas (AtualizaDescricaoService) e outro responsável por buscar no servidor *web* rotas que estejam aguardando para *download* (VerificaDownloads).

O serviço AtualizaDescriçãoService, possui um método interno que, fazendo uso da API Geocoder da Google, retorna as descrições de uma dada coordenada. Ainda existe um segundo método onde são calculadas as distâncias percorridas das rotas. A Listagem 7 apresenta o Geocoder API para a busca das descrições das rotas.

Listagem 7 – Geocoder API

```
Geocoder con = new Geocoder(context, Locale.getDefault());
try {
    List<Address> enderecos = con.getFromLocation(latitude,
longitude, 1);
    Address local = enderecos.get(0);
    retorno = local.getAddressLine(0);
} catch (Exception e) {
   e.printStackTrace();
}
```

Fonte: autoria própria.

O serviço VerificaDownloads é o responsável por fazer uma requisição ao servidor *web service* enviando o id do usuário autenticado, solicitando as rotas que esse usuário deseja baixar. A requisição é feita utilizando Json e a resposta, da mesma forma, retorna em Json. Para facilitar a utilização da API do *web service*, o formato do Json de requisição é o mesmo do Json de resposta para qualquer requisição. A Listagem 8 apresenta um Json de requisição de *download* e a Listagem 9 apresenta o Json de resposta com as rotas baixadas.

Listagem 8 – JSON de requisição de download de rotas

```
{
    "uuid_User": "ID-ÚNICO-DE-IDENTIFICACAO-DO-USUARIO-NO-
GOOGLE",
    "rotas": [ ],
    "msgObs": "verificaDownload"
}
```

Fonte: autoria própria.

Listagem 9 – JSON de resposta de *download* de rotas

```
{
    "uuid_User": "ID-ÚNICO-DE-IDENTIFICACAO-DO-USUARIO-NO-
GOOGLE",
    "rotas": [ {
            "rota": {
                "idWeb": 12,
                "idApp": 2,
                "uidUser": "ID-ÚNICO-DE-IDENTIFICACAO-DO-
USUARIO-NO-GOOGLE",
                "autor": "Cleverson Machado",
                "nomeRota": "Rota 45",
                "distancia": 67,
                "atualiza": false
            }, "pontos": [ {
                    "idWeb": 7,
                    "idRota": 6,
                    "idApp": 1,
                    "uidUser": "ID-ÚNICO-DE-IDENTIFICACAO-DO-
```

```
USUARIO-NO-GOOGLE",
                    "latitude": -25.83461,
                    "longitude": -52.7271,
                    "altitude": 470,
                    "dataHora": "26/06/2019 17:00",
                    "precisao": 14,
                    "velocidade": 1,
                    "descLocalizacao": "descrição do ponto",
                    "imagem": null
                }, {
                    "idWeb": 8,
                    "idRota": 6,
                    "idApp": 2,
                    "uidUser": "ID-ÚNICO-DE-IDENTIFICACAO-DO-
USUARIO-NO-GOOGLE",
                    "latitude": -25.8347154,
                    "longitude": -52.7249199,
                    "altitude": 0,
                    "dataHora": "26/06/2019 17:24",
                    "precisao": 10,
                    "velocidade": 12,
                    "descLocalizacao": "descrição do ponto",
                    "imagem": null
                }
            ]
        }
    ], "msgObs": "200"
}
```

Fonte: autoria própria.

Ressalta-se que, tanto o Json de requisição quanto o de resposta, seguem o modelo proposto da tabela Resposta apresentado no diagrama da Figura 4. Isso facilita o tratamento dos dados, já que tudo fica padronizado em um único modelo. Esse mesmo padrão também é utilizado nas requisições para postar uma rota, que será visto mais adiante.

Na Listagem 10 está o método responsável pela comunicação com o *web service*. O método utilizado é o mesmo para todas as requisições, pois o objeto enviado é sempre um JSON tendo como destino sempre o mesmo endereço.

Listagem 10 – Método de consumo do *web service*

```
Resposta resposta = null;
String UrlBase = "http://127.0.0.1:9991/publico";

try {
    RestTemplate restTemplate = new RestTemplate();
    restTemplate.getMessageConverters().add(new
MappingJackson2HttpMessageConverter());

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
```

```
    HttpEntity<Resposta> entity = new HttpEntity<>( enviar, headers
);
    ResponseEntity<Resposta> result =
restTemplate.postForEntity(UrlBase, entity, Resposta.class);

    try{
        resposta = result.getBody();
        return resposta;
    }catch (Exception ee){
        ee.printStackTrace();
        resposta = null;
    }
}catch (Exception e){
    e.printStackTrace();
}
```

Fonte: autoria própria.

Na Listagem 10 está exemplificado o modelo de comunicação com o *web service*. É utilizado o RestTemplate da Spring Framework pela facilidade de uso e pela padronização, já que no *web service* também será utilizado o *framework* da Spring. Para que o restTemplate funcione é preciso adicionar a dependência do spring-android-rest-template ao *gradle*, também é necessário adicionar ao *manifest* a permissão de Internet.

A seguir descreve-se como funciona a conexão com a base de dados SQLite utilizando como exemplo, a leitura dos dados das rotas para o cálculo das distâncias por um dos serviços citados anteriormente.

Ao abrir a aplicação *mobile*, ainda na tela de *Splash*, é instanciada a conexão com o banco de dados que será utilizada em todo o tempo de vida da aplicação. Na Listagem 11 está a instanciação que será única. Basicamente o que ocorre no método initializeInstanse é a criação de um objeto DbHelper que estende SQLiteOpenHelper. A criação dessa instância requer um contexto que, no caso será a aplicação, um nome de banco e uma versão de banco, controlando assim possíveis atualizações de base de dados.

Listagem 11 – Instancia responsável pela comunicação com o banco SqLite

```
DatabaseManager.initializeInstance(new DbHelper(this));
```

Fonte: autoria própria.

Com a instância de conexão com o banco de dados já criado, quando for necessária uma leitura aos dados basta apenas instanciar um objeto do tipo BD passando por parâmetro, no construtor, qual o tipo de acesso é necessário, ‘WR’ (de *Writable*) quando se precisa gravar dados na base de dados, ou ‘RD’ (de *Readable*) quando o acesso aos dados for somente

leitura, controlando, dessa forma, as concorrências internas por *lock* de dados.

Com o objeto BD criado, basta fazer uso de um de seus métodos definidos, no caso da recuperação de pontos de uma rota basta apenas utilizar o método recuperaPontosRota que recebe por parâmetro um filtro *where* do tipo texto, tornando assim a consulta genérica e flexível as diversas utilizações, e um parâmetro lágico que definirá se o campo imagem do tipo *byte* deve ser convertido para Base64 ou não. Essa conversão será útil e necessária nas leituras de dados para geração do Json para envio ao *web service*. O método recuperaPontosRota está exemplificado na Listagem 12.

Listagem 12 – Exemplo de método de leitura de dados SQLite

```
public ArrayList<Ponto> recuperaPontosRota(String where, boolean
converte){
        ArrayList<Ponto> retorno = null;
        try {
            cursor = database.rawQuery("select * from Ponto " +
where + ";", null);
            retorno = new ArrayList<Ponto>();
            cursor.moveToFirst();
            Ponto rs  = null;
            while (cursor.isAfterLast() == false) {
                rs = new Ponto();
rs.setIdApp(cursor.getInt(cursor.getColumnIndex("id_App")));
rs.setIdWeb(cursor.getInt(cursor.getColumnIndex("id_WEB")))
rs.setIdRota(cursor.getInt(cursor.getColumnIndex("id_Rota")));
rs.setUidUser(cursor.getString(cursor.getColumnIndex("Uid_user")));
rs.setLatitude(cursor.getDouble(cursor.getColumnIndex("latitude")));
rs.setLongitude(cursor.getDouble(cursor.getColumnIndex("longitude"))
);
rs.setAltitude(cursor.getDouble(cursor.getColumnIndex("altitude")));
rs.setVelocidade(cursor.getFloat(cursor.getColumnIndex("velocidade")
));
rs.setDataHora(cursor.getString(cursor.getColumnIndex("dataHora")));
rs.setPrecisao(cursor.getFloat(cursor.getColumnIndex("precisao")));
rs.setDescLocalizacao(cursor.getString(cursor.getColumnIndex("descLo
calizacao")));
rs.setImagem(cursor.getBlob(cursor.getColumnIndex("imagem")));
                rs.setImagemByteStr("");

                if(converte &&
                   rs.getImagem() != null
                ){
                    Deflater compressor = new Deflater();
                    compressor.setLevel(Deflater.BEST_COMPRESSION);
                    compressor.setInput(rs.getImagem());
                    compressor.finish();
                    ByteArrayOutputStream stream = new
ByteArrayOutputStream(rs.getImagem().length);

                    byte[] buf = new byte[1024];
```

```
                    while (!compressor.finished()) {
                        int count = compressor.deflate(buf);
                        stream.write(buf, 0, count);
                    }
                    try {
                        stream.close();
rs.setImagemByteStr(Base64.getEncoder().encodeToString(stream.toByte
Array()));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    rs.setImagem(null); // mando vazia para nao dar
exceção
                }
                retorno.add(rs);
                cursor.moveToNext();
            }
        }catch (Exception e){
            Log.e("erro  (BD)","Problema ao recuperar dados do banco
\n" + e.getMessage());
            e.printStackTrace();
        }finally {
            if(cursor != null)
               cursor.close();
        }
        return retorno;
    }
```

Fonte: autoria própria.

Com os dados dos pontos da rota carregados em memória, basta percorrê-los e a cada registro chamar o método calcDistancia que recebe por parâmetros duas coordenadas e retorna a distância entre essas duas coordenadas. Na Listagem 13 está apresentado o método de cálculo de distâncias.

Listagem 13 – cálculo da distância entre dois pontos de GPS

```
private Double calcDistancia(LatLng a, LatLng b){
    double distance = 0;
    Location location = new Location("posicao 1");
    location.setLatitude(a.latitude);
    location.setLongitude(a.longitude);
    Location location2 = new Location("posicao 2");
    location2.setLatitude(b.latitude);
    location2.setLongitude(b.longitude);
    distance = location.distanceTo(location2);
    return distance;
}
```

Fonte: autoria própria.

Finalmente após todas as distâncias calculadas, é preciso salvar essa informação na

rota específica. Para isso basta instanciar um objeto BD passando em seu construtor o parâmetro 'WR' e depois utilizar o método chamado atualizarRota que recebe via parâmetro uma rota. Na Listagem 14 está apresentado o método atualizarRota.

Listagem 14 – Exemplo de método de atualização de dados SQLite

```
public void atualizarRota(Rota rs){
  if(rs.getIdApp() > 0) {
  try {
     ContentValues contentValues = new ContentValues();
     contentValues.put("distancia", rs.getDistancia().toString());
     contentValues.put("atualiza", rs.getAtualiza());
     contentValues.put("id_WEB", rs.getIdWeb());

if
(rs.getUidUser().equals(UserAutenticate.getUsuarioLogado().getUuID()
));
   contentValues.put("exportado", "no");

database.update("Rota", contentValues, " id_App = ? AND Uid_user =
?", new String[]{"" + rs.getIdApp(), rs.getUidUser()});
      } catch (Exception e) {
        Log.e("ERRO (BD)", "pROBLEMA AO ATUALIZAR Rota!! \n" +
e.getMessage());
      }
  }
  else{
     try {
         incereRota(rs);
     }catch (Exception e){
         e.printStackTrace();
     }
  }
}
```

Fonte: autoria própria.

O método da Listagem 14 valida por meio de um *if* se a rota possui um id valorizado e, caso não possua, então o método executado é o de inserção de uma nova rota na base de dados. Na Listagem 15 está apresentado um exemplo de método de inserção na base de dados SQLite.

Listagem 15 – Exemplo de método de inserção de dados SQLite

```
public boolean incereRota(Rota rota) throws Exception {
        ContentValues contentValues = new ContentValues();
        contentValues.put("id_WEB", rota.getIdWeb());
        contentValues.put("Uid_user", rota.getUidUser());
        contentValues.put("Autor",    rota.getAutor());
        contentValues.put("nomeRota", rota.getNomeRota());
```

```
        contentValues.put("distancia", rota.getDistancia());
        contentValues.put("atualiza", rota.getAtualiza());
        contentValues.put("exportado", "no");
        if
(!rota.getUidUser().equals(UserAutenticate.getUsuarioLogado().getUuI
D()))
            contentValues.put("exportado", "yes");

        database.insert("Rota", null, contentValues);
        return true;
    }
```

Fonte: autoria própria.

Nas Listagens 13, 14 e 15 estão exemplos de três dos quatro tipos de acesso dados, que são, inclusão, alteração e leitura de dados, faltando apenas a exclusão que está exemplificada na Listagem 16. Esse método recebe via parâmetro o ID da rota a qual pertencem os pontos a serem excluídos, com esse ID é gerado um filtro *where* chamado, nesse caso de *query* e logo depois executado o método *delete* passando a tabela e a *query* de filtro.

Listagem 16 – Exemplo de método de exclusão de dados SQLite

```
public boolean excluirPontosRota(int _ID){
        String query = "";

        if (_ID > 0){
            query += " id_Rota = " + _ID;
        }
        else query = "";

        try {
            database.delete("Ponto", query, null) ;
        }catch (Exception e){
            Log.e("ERRO", "Problema ao deletar pontos da Rota !!
/n" + e.getMessage());
            return false;
        }
        return true;
    }
```

Fonte: autoria própria.

Com as Listagens 12, 14, 15 e 16 ficam exemplificadas todos os tipos de acesso aos dados utilizados no aplicativo *mobile*, nas leituras de rotas para exibição em mapa, listagens e compartilhamento os métodos utilizados serão os mesmos tornando-os assim genéricos e flexíveis a cada tipo de necessidade.

Voltando ao menu inicial da aplicação, o usuário possui algumas opções disponíveis, a primeira delas é iniciar um rastreamento ou continuar o último. Caso o usuário opte por

iniciar um novo rastreamento, será exigido que o ele informe um nome para a nova rota e também qual o meio utilizado para percorrer a rota, após informado os dados solicitados e clicado em criar o sistema validará as permissões de acesso ao sistema de GPS do dispositivo e caso não possua permissões suficientes, será solicitado ao usuário que ele dê permissões ao aplicativo para tal. Também é necessário que no arquivo androidManifest estejam inclusas as permissões ACCESS_FINE_LOCATION e ACCESS_COARSE_LOCATION. Com as permissões validadas o sistema instanciará um objeto do tipo rota e por meio de um objeto BD utilizará o método incereRota para salvar a rota na base de dados. Após salvar a rota, será disparado, um serviço responsável pela captura da posição geográfica atual do usuário. Essa captura ocorrerá em intervalos de tempos configuráveis pelo usuário na tela de configuração. Após realizada a captura, o sistema validará a qualidade do sinal que detectou a coordenada. Essa qualidade de sinal também é configurável, sendo que quanto maior seu valor, pior a qualidade de sinal. Estando a qualidade satisfatória, o sistema verificará se o usuário também configurou uma distância mínima nas preferências e , caso essa distância esteja configurada, o sistema, havendo um ponto anterior capturado, utilizará o método de cálculo de distâncias, apresentado na Listagem 13, verificará se a distância mínima foi atingida e, caso a resposta seja positiva, utilizará o método atualizarPontosRota da classe BD que fara a inclusão ou, caso já exista, alteração do ponto na rota que se esta sendo percorrida. Na Listagem 17 é apresentado o serviço de captura de rotas descrito anteriormente.

Listagem 17 – Serviço de captura de rotas

```
LocationManager locationManager = (LocationManager)
getApplicationContext().getSystemService(LOCATION_SERVICE);
boolean isGPSEnable =
locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
        boolean isNetworkEnable =
locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);

        if (!isGPSEnable && !isNetworkEnable) {
           Log.e(TAG, "nao ah GPS nem INTERNET disponível para
captura dos dados!! \n parando servicos ...");
           paraRastreamento();
        } else {
            if (isGPSEnable) {
                if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
                        ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
                    return;
                }
```

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
intervaloBusca, distanciaDesloc, this);
                if (locationManager != null) {
                    location =
locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
                    if (location != null) {
                        //Log.e("GPS latitude",
location.getLatitude() + "");
                        //Log.e("GPS longitude",
location.getLongitude() + "");
                        processaPonto(location);
                    }
                }
            }
            else if (isNetworkEnable ) {
                if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
                        ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
                    return;
                }
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVI
DER, intervaloBusca, distanciaDesloc, this);
                if (locationManager!=null){
                    location =
locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDE
R);
                    if (location != null){
                        processaPonto(location);
                    }
                }
            }
        }

private void processaPonto(Location location){
   if(LatAnte == 0 && LngAnte == 0) {
     LatAnte = location.getLatitude();
     LngAnte = location.getLongitude();
     return;
   }
   if(location.getAccuracy() > precisao)
      return;

   if(LatAnte == location.getLatitude() &&
      LngAnte == location.getLongitude() )
   return;
   else{
      LatLng pontoante = new LatLng(LatAnte, LngAnte);
      LatLng pontoAtual = new LatLng(location.getLatitude(),
location.getLongitude());

      if(distanciaDesloc > 0 &&
         Utils.calcDistancia(pontoante, pontoAtual) <
distanciaDesloc)
```

```
      return;
      }

      LatAnte = location.getLatitude();
      LngAnte = location.getLongitude();
      new FinalizaPonto(location).execute();
}
```

Fonte: autoria própria.

Na Listagem 18 está apresentado o método responsável por carregar as preferências do usuário.

Listagem 18 – Carregando preferências

```
private void buscaPreferencias() {
  rastreamentoativo =
UserPreferences.getPreferences().getBoolean("GPS_Status", false);
 idRota = UserPreferences.getPreferences().getInt("ultimaRota", 0);
 precisao =
Double.parseDouble(UserPreferences.getPreferences().getString("preci
saoGPS", "30"));
distanciaDesloc =
Float.parseFloat(UserPreferences.getPreferences().getString("distanc
iaIntervaloCaptura", "15"));
intervaloBusca =
Long.parseLong(UserPreferences.getPreferences().getString("tempoInte
rvaloCaptura", "5"));
intervaloBusca    = intervaloBusca * 1000;
}
```

Fonte: autoria própria.

Ainda na tela principal o usuário tem acesso a duas listas de rotas salvas no dispositivo, uma de rotas próprias e outra de rotas baixadas. Em resumo, a listagem exibida é a mesma mudando apenas o filtro de leitura das rotas na base de dados, pois dentro da rota existe o campo do id do usuário proprietário da rota e com o id do usuário autenticado é montado o filtro lendo as rotas com id igual ou diferente do id do usuário autenticado.

A listagem exibida faz uso de um *adapter* personalizado, semelhante ao do menu principal, também desenvolvido especialmente para a aplicação. A construção de um *adapter* personalizado esta exemplificada na Listagem 6.

Nas listas de rotas o usuário possui três opções sendo elas: visualizar a rota em um mapa, compartilhar a rota (na listagem de rotas baixadas essa opção fica desabilitada), e excluir a rota, sendo que essa exclusão somente ocorre para a rota salva no aplicativo. Rotas compartilhadas também devem ser excluídas pelo gerenciador *web*.

Ao clicar na opção para visualizar rotas a tela com o mapa é aberta utilizando uma

intent. O código para essa opção é apresentado na Listagem 19.

Listagem 19 – Exemplo de abertura de tela com *intent*

```
private void ver(int position) {
        Intent i = new Intent(context, MapaActivity.class);
        i.putExtra("id_Rota", getItemId(position));
        context.startActivity(i);
    }
```

Fonte: autoria própria.

Quando a tela de apresentação do mapa é aberta, o parâmetro da Intent é recuperado e com o ID da rota, os seus pontos são carregados do banco de dados com o uso de uma rotina semelhante à apresentada na Listagem 12. Para que o mapa possa ser exibido é preciso que a *activity* possua um componente visual do tipo Map e um objeto do tipo GoogleMap, além de implementar OnMapReadyCallback. Esse, por sua vez, obriga a implementação do método onMapaReady(), que é o responsável por atribuir o mapa ao componente de tela e atribuir as configurações do mapa. Na Listagem 20 está o método onMapaReady().

Listagem 20 – Método onMapaReady()

```
@UiThread
public void onMapReady(GoogleMap googleMap) {
    mMap          = googleMap;
    mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);

    mMap.setOnMarkerClickListener(new
GoogleMap.OnMarkerClickListener(){
    public boolean onMarkerClick(Marker marker){
        pos = marker.getPosition();
        return false;
    }
});
mMap.setInfoWindowAdapter(new GoogleMap.InfoWindowAdapter() {
    @Override
    public View getInfoWindow(Marker marker) {
        return null;
    }
    @Override
    public View getInfoContents(Marker marker) {
        TextView tvTexto = new TextView(RotasActivity.this);
        tvTexto.setText(Html.fromHtml("<b><font color=\"#ff0000\">"
+ marker.getTitle() + "</font> </b> <br>" + marker.getSnippet() ));
        return tvTexto;
    }
});
}
```

Fonte: autoria própria.

Após a execução do método onMapaReady, o trajeto percorrido na rota é inserido no mapa e, caso o usuário configure nas preferências, também são inseridos no mapa os marcadores de pontos com algumas informações sobre o local do marcador. Também nessa rotina são inseridas as imagens vinculadas aos pontos, caso existam. A Listagem 21 apresenta a rotina de inserção desses dados no mapa para visualização.

Listagem 21 – Inserindo pontos e linhas no mapa

```
@UiThread
private void adicionaPontoMapa(Ponto ponto) {
   posicao = new LatLng(ponto.getLatitude(), ponto.getLongitude());
   final MarkerOptions op;
   if(!repercorrendo) {
     if (altitudeAnte == 0) {
        altitudeAnte = ponto.getAltitude();
        declive = 0;
        elevacao = 0;
     } else {
       if (altitudeAnte > ponto.getAltitude()) {
          declive += altitudeAnte - ponto.getAltitude();
     } else {
          elevacao += ponto.getAltitude() - altitudeAnte;
     }

     altitudeAnte = ponto.getAltitude();
  }

  op = new MarkerOptions().position(posicao)
      .title("Dados do ponto selecionado.")
      .snippet("<br><br>Data / Hora : " + ponto.getDataHora() +
      "<br>Altitude : " + decFormat.format(ponto.getAltitude()) + "
Mts."+
      "<br>Local : " + ponto.getDescLocalizacao() +
      "<br>Velocidade : " + decFormat.format(ponto.getVelocidade())
+
      " Km/h." +
      "<br><br>GANHOS:" +
      "<br>Elevação:" + decFormat.format(elevacao) + " Mts" +
      "<br>Declive:" + decFormat.format(declive) + " Mts"
      );

  if(ponto.getImagem() !=null) {
    Bitmap bmp = BitmapFactory.decodeByteArray(ponto.getImagem(), 0,
ponto.getImagem().length);

op.icon(BitmapDescriptorFactory.fromBitmap(criaMarcador(MapaActivity
.this, bmp, "Nupur")));
  }
  }else{
    op = new MarkerOptions().position(posicao)
```

```
          .title("Dados do ponto.")
          .snippet("<br><br>" +
          "<br>Altitude : " + decFormat.format(ponto.getAltitude()) +
          " Mts." + "<br>Velocidade : " +
decFormat.format(ponto.getVelocidade()) + " Km/h.")
.icon(BitmapDescriptorFactory.fromResource(R.drawable.mark1));

    if (lastPosition != null) {
       Polyline line = mMap.addPolyline(new PolylineOptions()
       .add(lastPosition, posicao)
       .width(5)
       .color(Color.YELLOW));
    }
    lastPosition = posicao;
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(posicao,
zoom));
  }

  if(repercorrendo || verMarcadores || ponto.getImagem() !=null) {
    this.runOnUiThread(new Runnable() {
    @Override
      public void run() { mMap.addMarker(op); }
   });
  }
}
```

Fonte: autoria própria.

Ainda na Listagem 21 pode-se verificar a inserção de componentes do tipo *polyline*. Esses componentes são linhas que ligam dois pontos no mapa e ao final pode-se verificar a chamada para o método moveCamera, responsável por direcionar e centralizar o foco no último marcador inserido no mapa, aplicando um nível de *zoom* pré-configurado na tela de preferências.

Após a inserção do percurso no mapa, o aplicativo verifica, por meio da leitura de uma preferência, semelhante a da Listagem 18, se o rastreador está ativo. Significando, assim, que novos pontos serão capturados para a rota e esta deverá ser atualizada. Se o rastreador estiver ativo, o sistema remove a opção para repercorrer a rota e ativa a opção de fotografar, que está disponível na tela e inicia uma *thread* responsável por recuperar os novos pontos e atualizar o mapa.

Caso o rastreador esteja ativo e a rota visualizada seja a mesma que se está percorrendo, o usuário poderá incluir imagens a rota. Para isso, no mapa, está disponível um botão com a imagem de uma câmera (apresentado na Figura 14), ao ser clicado, o sistema, por meio do uso de uma *intent* semelhante ao da Listagem 19, abre o aplicativo de câmera do dispositivo (caso este possua), para que a imagem possa ser capturada. Após a imagem ser capturada, o sistema salva no banco de dados um ponto com a coordenada e a imagem.

O usuário ainda terá disponível no mapa um botão para bloquear a tela do dispositivo e assim evitar cliques indesejados enquanto estiver fazendo o percurso e visualizando o mapa (como mostrado na Figura 14).

Caso o rastreador não esteja ativo, então o usuário terá disponível na tela um botão 'Guiar', que ao ser selecionado iniciará uma rotina de captura de posições GPS, semelhante à rotina apresentada na Figura 17. A rotina irá capturar a posição do usuário e exibir marcadores diferenciados no mapa para que o usuário possa localizar-se e identificar qual direção deverá tomar para repercorrer a rota sendo exibida no mapa.

Voltando as opções disponíveis nas listagens de rotas, há a opção 'Excluir', que, ao ser clicado, exibe um alertDialog, solicitando confirmação de exclusão e, caso seja confirmado, a rotina de exclusão da rota será executada. E os pontos da rota também são excluídos por meio de uma rotina semelhante a que exclui o registro da rota. A Listagem 22 apresenta o componente AlertDialog.

Listagem 22 – Exclusão de rotas

```
AlertDialog.Builder builder = new AlertDialog.Builder(context);
builder.setTitle("Confirma Exclusão");
builder.setMessage("Deseja realmente excluir a rota selecionada
??");
builder.setPositiveButton("Sim", new
DialogInterface.OnClickListener() {
  public void onClick(DialogInterface arg0, int arg1) {
     new
DeletaRotaBase().execute(((Rota)getItem(position)).getIdApp());

     if( ((Rota) getItem(position)).getIdApp() ==
UserPreferences.getPreferences().getInt("idRotaAtual", 0)){
        SharedPreferences.Editor editor =
UserPreferences.getPreferences().edit();
                    editor.putBoolean("GPS_Status", false);
                    editor.commit();
     }
     rotas.remove(position);
     Toast.makeText(context , "Exclusão feita com sucesso!! ",
Toast.LENGTH_SHORT).show();
     atualizarLista(rotas);
  }
});
alerta = builder.create();
alerta.show();
}
```

Fonte: autoria própria.

Ainda nas opções disponíveis nas listagens de rotas, mais específicamente na listagem

de rotas próprias, o usuário tem a opção de compartilhar uma rota, ou seja, ele pode enviar essa rota para o aplicativo *web* para que os demais usuários da rede possam visualizar e baixá-la caso desejem.

O compartilhamento da rota consiste em carregar todos os dados da rota salvos na base SQLite (Listagem 12), adicionar essas informações em um objeto do tipo Resposta e a partir desse, gerar um arquivo do tipo Json, semelhante ao da Listagem 9 porém com o atributo msgObs valorizado com 'gravar'. A partir dai é feito a comunicação com o servidor (Listagem 10), e aguardado o retorno, que será o mesmo Json enviado, porém agora, os dados das rotas possuem um valor a mais que será o id único do registro na base de dados *web*. Dessa forma é possível identificar todos os dados que obtiveram êxito ao serem compartilhados, esses dados que tiveram sucesso no compartilhamento serão atualizados no SQLite recebendo o valor do id do registro no banco de dados *web* para que, em compartilhamentos futuros, não sejam gerados registros duplicados.

Voltando a tela principal, o usuário também tem a opção de visualizar a última rota percorrida. Ao ser selecionado essa opção, o mapa será aberto apresentando a rota e caso a rota esteja sendo percorrida no momento, novos pontos são atualizados. Tudo ocorre da mesma forma que na visualização de rota a partir da listagem de rotas descrita anteriormente.

Na listagem de opções da tela principal, também está a opção "Configurações" que, ao ser selecionada, exibe uma nova tela na qual podem ser realizadas algumas configurações do sistema. A nova tela também é aberta com o uso de um *intent* sem parâmetros. A *activity* aberta estende PreferenceActivity, que, por definição, salva em SharedPreference todos os dados que são informados nos campos apresentados usando um arquivo de leiaute. Esse arquivo de leiaute é diferenciado dos arquivos normais de outras *activities*. Ele utiliza como definidor de leiaute um PreferenceScreen e suas opções de configurações podem ser separadas em categorias por meio das Tags PreferenceCategory. A Listagem 23 apresenta o arquivo de leiaute de preferências.

Listagem 23 – *Layout* de preferências

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <PreferenceCategory
        android:title="Preferencias do Rastreador"
        android:key="pref_Maps">
        <EditTextPreference
```

```
        android:key="nivel_zoom"
        android:title="Zoom do mapa"
        android:inputType="number"
        android:defaultValue="14"/>
        <CheckBoxPreference
            android:key="ver_Markers"
            android:textIsSelectable="true"
            android:title="Ver Marcadores"
            android:checked="false"/>
        <EditTextPreference
            android:key="tempoIntervaloCaptura"
            android:title="Tempo de atualização GPS em Seg."
            android:inputType="number"
            android:defaultValue="2"/>
        <EditTextPreference
            android:key="distanciaIntervaloCaptura"
            android:title="Distância de atualização GPS em Mts."
            android:inputType="number"
            android:defaultValue="10"/>
        <EditTextPreference
            android:key="precisaoGPS"
            android:title="Precisão do GPS"
            android:inputType="number"
            android:defaultValue="10"/>
    </PreferenceCategory>
</PreferenceScreen>
```

Fonte: autoria própria.

A Listagem 24 apresenta novamente um exemplo de como as preferências podem ser recuperadas em outras *actiivities*.

Listagem 24 – Recuperando dados de preferências

```
SharedPreferences pref =
PreferenceManager.getDefaultSharedPreferences(getBaseContext());
RastreioAtivo = pref.getBoolean("GPS_Status", false);
Integer id    = pref.getInt("idRotaAtual",0);
```

Fonte: autoria própria.

Por fim, no menu principal está a opção de *logoff* do usuário que, ao ser clicada, finaliza qualquer rotina que esteja execução, finaliza a sessão autenticada do usuário e abre novamente a tela de *login* da aplicação. A Listagem 25 apresenta como é feito o encerramento da sessão do usuário e chamada da tela de *login* da aplicação.

Listagem 25 – Finalização da sessão do usuário autenticado

```
private void logoutUser() {
        atualizaTela = false;
        GoogleSignInClient gsc =
UserAutenticate.getGoogleSignInClient();
        gsc.signOut().addOnCompleteListener(this, new
```

```
OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull Task<Void>
task) {
    login();
                    }
                });
    }
    private void login() {
        UserAutenticate.logout();
        intent = new Intent(MainActivity.this,
LoginActivity.class);
        startActivity(intent);
        finish();
    }
```

Fonte: autoria própria.

3.4.2 Implementação do aplicativo *web service*

A implementação do *web service* faz uso do *framework* Spring para gerir seu funcionamento. Na parte de segurança o Spring Security é o responsável pela autenticação dos usuários *web* e controles de acesso. Na parte de acesso a dados, o Spring JPA é o responsável pelo gerenciamento de transações com o banco de dados. Tudo isso realizado por meio do Spring Boot, um pacote de ferramentas que facilitam e agilizam o desenvolvimento de aplicações que utilizam o Spring.

Ao adicionar a dependência do Spring Security à aplicação, todo o acesso sem *token* de autorização fica bloqueado, porém, dessa forma, não seria possível que o primeiro acesso ao serviço para fazer o *login* e gerar o *token* fosse feito. Para autorizar esse acesso, o Spring Security possui uma classe de configuração na qual se pode especificar quais URLs do *service* necessitam de restrição e quais URLs não necessitam. A Listagem 26 apresenta essa classe de configuração. Neste exemplo os acessos as URLs a partir de '/login' e '/publico' possuem acesso sem necessitarem de *token* de autenticação.

Listagem 26 – Exemplo de *repository*

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter{
      @Autowired
      private UserDetailsServiceImpl userService;
      @Autowired
      private JwtAuthEntryPoint errorHandler;
      @Bean
```

```
	public JwtAuthFilter authTokenFilter() {
		return new JwtAuthFilter();
	}
	@Bean
	public PasswordEncoder passwordEncoder() {
		return new BCryptPasswordEncoder();
	}
	@Bean
	@Override
public AuthenticationManager authenticationManagerBean() throws
Exception {
	return super.authenticationManagerBean();
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
	auth.userDetailsService(userService).passwordEncoder(passwordEn
coder());
}

@Override
protected void configure(HttpSecurity http) throws Exception {
http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy
.STATELESS);
	http.cors().and().csrf().disable()
	    .authorizeRequests()
	    .antMatchers("/login/**", "/publico/**").permitAll()
	    .anyRequest().authenticated()
	    .and()
	.exceptionHandling().authenticationEntryPoint(errorHandler);

	http.addFilterBefore(authTokenFilter(),
UsernamePasswordAuthenticationFilter.class);
	}
@Bean
public CorsConfigurationSource corsConfigurationSource() {
	UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
  CorsConfiguration conf = new
CorsConfiguration().applyPermitDefaultValues();
  conf.addAllowedOrigin("*");
  source.registerCorsConfiguration("/**", conf);
  return source;
}
```

Fonte: autoria própria.

O aplicativo de *web service* conta com três ‘portas’ de comunicação externa, uma responsável pela autenticação de usuários da aplicação *web*, a segunda responsável pela comunicação de dados com a aplicação *web* e a terceira responsável pela comunicação de dados com a aplicação *mobile*.

A ‘porta’ responsável pela autenticação dos dados de *login* dos usuários *web* possui um método que recebe um parâmetro do tipo Usuario. Esse parâmetro possui as informações dos dados do usuário retornado pela API Google Auth após o *login* no aplicativo *web*. Este, ao receber essas informações, faz uma busca no banco de dados, utilizando um *repository* de Usuario, para identificar se esse usuário já existe na base de dados local, caso exista, as informações são atualizadas com as informações recebidas via parâmetro, se o usuário não existir na base de dados local um novo registro é criado.

Na Listagem 27 é apresentado o *repository* da tabela Usuario, ela é quem faz o interfaceamento entre a aplicação e o banco de dados por meio do JPA do Spring.

Listagem 27 – Exemplo de *repository*

```
@Repository
public interface UsuarioRepository extends
JpaRepository<Usuario, Integer>{
      Optional<Usuario> findByLogin(String login);
      Optional<Usuario> findByUuid(String uuid);
}
```

Fonte: autoria própria.

Após os dados do usuário serem gravados, o Spring Security faz a autenticação do *login* desse usuário no *web service* gerando um *token* de autorização com um determinado tempo de validade. Esse *token* é gerado utilizando um usuário e uma chave tendo uma data de inicio e fim de validade. Com o *token* gerado, o Spring autoriza que este usuário possa acessar as outras URLs de comunicação. A Listagem 28 apresenta a classe JWT responsável por gerar o *token* de acesso, validar se um *token* recebido é valido e extrair do *token* o usuário autenticado com tal *token*. Esses dois últimos métodos são utilizados nas requisições de dados feitas pelos usuários autenticados.

Listagem 28 – Classe JWT de geração de *token* de acesso

```
@Component
public class JwtProvider {
      private static final String SECRET_KEY = "pos-java";
      public String generateToken(Authentication auth) {
            UserAuthDetails user = (UserAuthDetails)
auth.getPrincipal();
      Date agora = new Date();
      Date exp = new Date(agora.getTime() + (60 * 60 * 24 * 1000));
      return Jwts.builder()
            .setSubject(user.getUsername())
            .setIssuedAt(agora)
            .setExpiration(exp)
```

```
            .signWith(SignatureAlgorithm.HS512, SECRET_KEY)
            .compact();
    }

    public String getUserFromToken(String token) {
       return Jwts.parser()
            .setSigningKey(SECRET_KEY)
.parseClaimsJws(token)
            .getBody().getSubject();
    }

    public boolean validateToken(String token) {
     try {
       Jwts.parser()
       .setSigningKey(SECRET_KEY)
       .parseClaimsJws(token);
     } catch(Exception e) {
       System.out.println("Token inválido! " + e.getMessage());
       return false;
     }
     return true;
    }
}
```

Fonte: autoria própria.

Na Listagem 29 é apresentado o método geraLogin(), responsável por receber os dados do usuário e fazer as devidas atualizações de dados e devolver ao seu cliente as informações do usuário autenticado com um *token* para próximas requisições.

Listagem 29 – Método de autenticação do sistema

```
@PostMapping
@RequestMapping("/saveLoginUser")
public ResponseEntity<RetornoLoginDTO> geraLogin(@RequestBody
Usuario user) {
    Usuario usuario = new Usuario();
    usuario = userReposit.findByLogin(user.getLogin()).orElse(null);

    if(usuario == null) {
       usuario = new Usuario();
       usuario.setId(0);
       List<Papel> papeis = new ArrayList<>();
       papeis.add(new Papel("ROLE_USER"));
       usuario.setPapeis(papeis);
    }
    usuario.setLogin(user.getLogin());
    usuario.setNome(user.getNome());
    usuario.setUuid(user.getUuid());
    BCryptPasswordEncoder enc = new BCryptPasswordEncoder();
usuario.setSenha(enc.encode(user.getSenha()));
    try{
       userReposit.save(usuario);
```

```
      Authentication auth = authManager.authenticate( new
UsernamePasswordAuthenticationToken(usuario.getLogin(),user.getSenha
()));

      SecurityContextHolder.getContext().setAuthentication(auth);
      String token = jwtProvider.generateToken(auth);
      UserAuthDetails details = (UserAuthDetails)
auth.getPrincipal();
      RetornoLoginDTO retorno = new RetornoLoginDTO();
      etorno.setNome(details.getNome());
      retorno.setId(details.getId());
      retorno.setToken(token);
      retorno.setUuid(details.getUuid());
      List<String> papeis = details.getAuthorities().stream().map(g -
> g.getAuthority()).collect(Collectors.toList());
      retorno.setPapeis(papeis);
      return ResponseEntity.ok(retorno);
    }catch(Exception e){
      e.printStackTrace();
    }
    return null;
}
```

Fonte: autoria própria.

Ressalta-se que se, futuramente, for de interesse que o sistema possua um sistema de *login* próprio, basta apenas que se mantenha nesse método a parte de autenticação dos dados recebidos e o restante do sistema funcionará devidamente.

A segunda ‘porta’ de acesso do *web service* é a que possibilita que usuários da aplicação *web* façam requisições de dados ao *web service*. Essa ‘porta' possui apenas um único método que recebe como parâmetro um objeto Json do tipo Resposta. Todas as requisições de dados de rotas, exclusões, inclusões de comentários são recebidos por essa ‘porta’. No Json recebido há um atributo que indica qual funcionalidade dentro do *web service* deve ser executada de acordo com seu valor. Na Listagem 30 e apresentada a classe do tipo *controller*.

Listagem 30 – Classe *controller* de acesso a dados *web*

```
@RestController
@RequestMapping("/rotas")
public class RotaController {
      @Autowired
      private RotaRepository reposit;
      @Autowired
      private PontoRepository pontoReposit;
      @Autowired
      private UsuarioRepository userReposit;
      @Autowired
```

```
      private DownloadRepository downloadReposit;
      @Autowired
      private ComentarioRepository comentarioReposit;
      private String uuid;
      private Resposta resposta;
      private List<RotaAdapter> adapters;
@PostMapping
@RequestMapping("/interfaceWEB")
public ResponseEntity<Resposta> trataSolicitacao(@RequestBody
Resposta requisicao){
  this.uuid = requisicao.getUuId();
  if(!validaRequisicao()) {
  requisicao.setMsg("403");
  new Exception("Acesso nao AUTORIZADO ao serviço!! Faça
login");
  return ResponseEntity.ok(requisicao);
}
resposta = new Resposta();
adapters = new ArrayList<>();
switch (requisicao.getMsgObs()){
case "getLocations":
resposta = recuperaPontos(requisicao);
break;
case "getProprias":
resposta = listar("proprias");
break;
case "getOutros":
resposta = listar("outros");
break;
case "deletaRota":
resposta = deletaRota(requisicao);
break;
case "baixar":
resposta = agendaDownloadRota(requisicao);
break;
case "incluirComentario":
resposta = incComentario(requisicao);
break;
case "excluirComentario":
resposta = deletaComentario(requisicao);
break;
default:
resposta.setUuId(this.uuid);
resposta.setMsgObs("400");
break;
}
return ResponseEntity.ok(resposta);
}
```

Fonte: autoria própria.

Perceba que toda a requisição que chegar até o servidor *web* com um objeto de entrada valido, receberá como retorno um objeto do mesmo tipo, contendo no campo msgObs o *status* do processamento da solicitação, sendo 400 para método não encontrado ou erro no

processamento dos dados e 200 para execução ok. Também receberá neste mesmo objeto os dados solicitados.

Na Listagem 31 está um dos métodos internos chamados o agendarDownloadRota. Este método recebe como parâmetro um objeto do tipo resposta contendo o id do usuário solicitante do *download* e a rota que ele deseja baixar além do campo msgObs valorizado com 'baixar', indicando ao *web service* que é uma solicitação de download de rota.

Listagem 31 - exemplo de método interno da aplicação *web service* para conexão *web*

```
private Resposta agendaDownloadRota(Resposta requisicao) {

   Resposta resp = new Resposta();
   resp.setUuId(this.uuid);
   resp.setMsgObs("200");
   try {
   Download download = new Download();
   download.setUidUser(requisicao.getUuId());
   download.setRota(new
Rota(requisicao.getRotaAdapter().get(0).getRota().getIdWeb()));
   if(downloadReposit.findByRotaAndUidUser(download.getRota(),
download.getUidUser()).orElse(null) == null) {
      downloadReposit.save(download);
   }
   }catch (Exception e) {
      resp.setMsg("400");
      e.printStackTrace();
   }
   return resp;
}
```

Fonte: autoria própria.

Os demais métodos também funcionam da mesma forma que o método anterior, instanciando a Classe Resposta e as demais classes necessárias de acordo com a resposta esperada pelo cliente.

O terceiro método, responsável por receber e tratar as requisições do aplicativo *mobile*, de forma muito semelhante ao método anterior, também possui apenas um método de acesso que recebe via parâmetro um Json do objeto Resposta e, validando o conteúdo do atributo msgObs entende qual método executar. Na Listagem 32 é apresentado o método de acesso.

Listagem 32 – Método utilizado para receber requisições da aplicação *mobile*

```
@Override
@RequestMapping("/publico")
@GetMapping(path= "/", consumes = "application/json", produces =
```

```
"application/json")
public ResponseEntity<Resposta> recebeRequisicao( @RequestBody
Resposta requisicao)  {
 Resposta resp = requisicao;
 String method = resp.getMsgObs();
 String uuid   = resp.getUuid_User();
 resp.setMsgObs("OK");
 boolean validado = validaUsuario(resp.getUuid_User());
 if(validado) {
   switch(method) {
     case "grava":
     resp = gravaRotas(resp);
     break;
     case "verificaDownload":
     resp = downloadRotas(resp);
     break;
     default:
     resp.setMsgObs("Opção Não identificada.");
     break;
   }
   return new ResponseEntity<Resposta>(resp, HttpStatus.OK);
 }
 else
   return new
ResponseEntity<>(HttpStatus.NON_AUTHORITATIVE_INFORMATION);
}
```

Fonte: autoria própria.

Na Listagem 33 está listado, como exemplo, o método interno responsável por buscar e devolver as rotas que um determinado usuário solicitou baixar.

Listagem 33 – Exemplo de método interno da aplicação *web service* para requisições *mobile*

```
private Resposta downloadRotas(Resposta requisicao) {
   Resposta resp = new Resposta();
   List<RotaAdapter> listAdp = new ArrayList<>();
   resp.setUuid_User(requisicao.getUuid_User());
   Rota rota = null;
   try {
    List<Download> downloads =
downloadRepository.findAllByUidUser(requisicao.getUuid_User());
  for(Download baixar : downloads) {
    RotaAdapter adp = new RotaAdapter();
    rota =
rotaRepository.findByIdWeb(baixar.getRota().getIdWeb()).orElse(null)
;
    if(rota == null) continue;
if(!rota.getUidUser().equalsIgnoreCase(requisicao.getUuid_User()))
       rota.setIdApp(0);

    adp.setRota(rota);
    List<Ponto> pontos = new ArrayList<>();
    List<Ponto> pts = new ArrayList<Ponto>();
```

```
    pts = pontoRepository.findAllByRota(rota);
    for( Ponto ponto : pts){
     if(!rota.getUidUser().equalsIgnoreCase(requisicao.getUuid_User(
)))          ponto.setIdApp(0);
      if(ponto.getImagem() != null) {
       try {

ponto.setImagemByteStr(Utils.encoderDados(ponto.getImagem()));
        } catch (Exception ee) {
           ee.printStackTrace();
        }
       }
     pontos.add(ponto);
    }
    adp.setPontos(pontos);
    listAdp.add(adp);
    downloadRepository.delete(baixar);
  }
  resp.setRotas(listAdp);
  }catch (Exception e) { }
  return resp;
}
```

Fonte: autoria própria.

3.4.3 Implementação do aplicativo *web*

Ao iniciar a aplicação *web* sem autenticação é apresentada a tela de *login*. Da mesma forma que na aplicação *mobile*, o único modo de *login* disponível é utilizando uma conta do Google, pois a autenticação também é via Google Auth.

Para que o *login* via API Google funcione, inicialmente é preciso fazer uma configuração no módulo principal da aplicação importando algumas bibliotecas necessárias. Na Listagem 34 é exibida essa configuração.

Listagem 34 – Importação de bibliotecas para uso da API de autenticação Google

```
Imports: [
    AngularFireModule.initializeApp(environment.firebase),
    AngularFirestoreModule,
    AngularFireAuthModule
      ]
```

Fonte: autoria própria.

A tela de login existe apenas um botão Para que o usuário possa autenticar-se no sistema. Ao clicar nesse botão, a aplicação chama a API do Google Auth que, solicitará os dados de *login* e efetuará a autenticação via AngularFireAuth, devolvendo como resultado um

objeto contendo os dados do usuário autenticado, da mesma forma que ocorre na aplicação *mobile,* descrita na Seção 4.4.1. Com esse retorno a aplicação valida se obteve sucesso ou erro na autenticação. Caso a autenticação tenha sido falha, uma mensagem é exibida e a aplicação ficará na tela de *login*. Caso a autenticação tenha ocorrido com sucesso, a aplicação *web* carrega os dados do usuário, retornado pelo Google, para um objeto do tipo Usuario e envia esse objeto para o *web service* para que esse usuário seja persistido na base de dados web e receba o *token* de autenticidade. A Listagem 35 apresenta o método de autenticação e os processamentos posteriores descritos.

Listagem 35 – Autenticação de usuário *web*

```
async login(){
    this.usuario = null
      new Promise<any>((processa) => {
      let provider = new auth.GoogleAuthProvider();
      provider.addScope('profile');
      provider.addScope('email');
      this.afAuth.auth.signInWithPopup(provider).then(res => {
        this.usuario = new Usuario()
        this.usuario.uuid = res.user.providerData[0].uid
        this.usuario.nome = res.user.providerData[0].displayName
        this.usuario.login = res.user.providerData[0].email
        this.usuario.senha = res.user.uid
        this.gravaLogin()
      }, error =>{
         this.msgs = [{
         severity: 'error', summary: 'Erro',
         detail: 'Falha ao efetuar login!! Tente Novamente ...'
        }];
        this.router.navigateByUrl("login")
      })
    })
  }
  gravaLogin(){
    this.http.post<RetornoLogin> ("https://localhost:9991/login/save
LoginUser"
, this.usuario).subscribe(
      ret => {
        this.auth.login(ret)
        this.router.navigateByUrl("/mywalking/home/rotas");
      },
      erro => {
       this.msgs = [{
         severity: 'error', summary: 'Erro',
         detail: 'Nao foi possivel efetuar o login!! Tente Novamente
 ...'
        }];
      }
   );
  }
```

Fonte: autoria própria.

Ressalta-se que o método chamado, para que a autenticação via Google seja executada, é chamado de signInWithPopup do AngularFireAuth, que espera como parâmetro um objeto Provider contendo as informações que deverão ser retornadas após a autenticação. Esse método também aceita funções de sucesso e falha que serão executadas após a autenticação, dessa forma, é passada uma função de sucesso que criará o objeto Usuario e fará a comunicação com o *web service* e, também, a função de erro que mostrará a mensagem de erro e chamará a tela *login* novamente.

Após o retorno do *login* no *web service*, a aplicação recupera o *token* de conexão de devera ser utilizado nas demais requisições ao *web service*. O *token* também é armazenado no objeto Usuario da sessão. Casso ocorra erro na autenticação com o *web service* também será exibida a mensagem de falha e voltará para a tela de *login*.

Se tudo ocorrer corretamente a aplicação abrirá a tela com as rotas que o usuário compartilhou, para isso, no método ngOnInit da lista de rotas do usuário, é disparada uma consulta ao *web service* solicitando as rotas que o usuário publicou. As informações do usuário são enviadas em um objeto do tipo Resposta que contém também, no campo msgObs o texto 'getProprias'.

A Listagem 36 apresenta o método que recupera as rotas compartilhadas pelo usuário para exibição na listagem de rotas.

Listagem 36 – Método responsável por disparar a consulta de rotas compartilhadas

```
this.map.getRotas("propria").subscribe(data => {
        if(data.rotaAdapter != null){
          data.rotaAdapter.forEach(rr =>
{this.rotas.push(rr.rota)})
        }
        else{
          this.rotas = []
          this.msgs = [{
            severity: 'error', summary: 'Erro',
            detail: 'Não ha Rotas para serem exibidas!!'
          }];
        } }
```

Fonte: autoria própria.

O método apresentado na Listagem 36 executa um segundo método que recebe via parâmetro quais os tipos de rotas desejadas, no caso, próprias. Esse segundo método existe, pois há a listagem de rotas postadas pelos outros usuários e, dessa forma, uma mesma

consulta poderá recuperar os dois tipos de rotas. A Listagem 37 apresenta o método getRotas e o método comunica que é o método principal em todas as comunicações com o *web service*.

Listagem 37 – Método responsável por disparar a consulta de rotas compartilhadas

```
getRotas(tipo){
    this.resposta = new Resposta()
    this.resposta.uuId = this.auth.getUuid()
    if(tipo == "propria")
       this.resposta.msgObs = "getProprias"
    else
       this.resposta.msgObs = "getOutros"
    return this.comunica()
  }

  comunica(){
    return
this.http.post<Resposta>("HTTPS://localhost:9991/rotas/interfaceWEB"
, this.resposta)
  }
```

Fonte: autoria própria.

Em nenhum momento foi adicionado o *token* do usuário à conexão para que o *web service* aceite a requisição de conexão da aplicação *web*. Isso não é necessário, pois existe um *interceptor* configurado que identifica qualquer conexão com a aplicação externa e adiciona automaticamente o *token* de autenticação ao cabeçalho da requisição. A Listagem 38 apresenta o *interceptor*.

Listagem 38 – *interceptor* de requisições web

```
export class AuthInterceptorService implements HttpInterceptor {
  constructor(private auth: AuthService) { }
  intercept(req : HttpRequest<any>,next : HttpHandler) :
Observable<HttpEvent<any>>{
    let token = this.auth.getToken()
    if(token != null){
       let authReq = req.clone({
          headers : req.headers.set("Authorization", "Bearer " +
token)
       })
       return next.handle(authReq)
    } else{
      return next.handle(req)
    }
  }
}
```

Fonte: autoria própria.

Sua configuração é bastante simples nas últimas versões do Angular, bastando apenas ter uma classe implementando HttpInterceptor, implementar o método *intercept* que, recebe via parâmetro, quais tipos de requisições serão interceptadas, no caso *any* (todas) e um controlador que permitirá a passagem do objeto de requisição adiante (HttpHandler), além de retornar a requisição já tratada (return next handle(req)). Com isso a classe *interceptor* é criada, agora só basta adicioná-la ao módulo principal da aplicação (app.module). A Listagem 39 mostra essa adição.

Listagem 39 – Adicionando interceptor ao módulo da aplicação

```
@NgModule({
  imports: [
    HttpClientModule
  ],
  providers: [ {
      provide:            HTTP_INTERCEPTORS,            useClass:
AuthInterceptorService, multi : true }
  ],
  bootstrap: [AppComponent]
})
```

Fonte: autoria própria.

Após a requisição ter sido tratada pelo *interceptor* ela é enviada ao *web service* para processamento e, se tudo ocorrer como o esperado, a lista de rotas compartilhadas será entregue à aplicação *web* em um formato JSON que será lido e convertido em objeto pela função de *call-back* passada por parâmetro para, nesse caso, o método getRotas.

Após o término do processamento, existirá uma lista de rotas prontas para serem exibidas pela aplicação *web* ao usuário, então, com a ajuda do Angular e do PrimeNG uma tabela é desenhada na tela contendo as informações das rotas e, no caso das rotas próprias, haverá três botões de opções: ver, baixar e excluir. No caso da listagem de rotas de outros usuários o processo é exatamente igual, exceto pela requisição enviada ao *web service* solicitar as rotas de terceiros e por não existir a opção de exclusão na listagem exibida.

Ao clicar na opção excluir, uma nova requisição é disparada ao *web service* contendo um JSON de resposta, porém, haverá uma rota nesse JSON e o valor do atributo msgObs será ‘deletaRota’. Dessa forma, o *web service* saberá que a requisição se trata de uma exclusão e saberá qual rota deverá excluir. Após a exclusão a aplicação *web* receberá como resposta um JSON do tipo resposta com o *status* do processo e, de acordo com o código recebido exibirá a *msg* de sucesso ou falha na exclusão. Todas as requisições ao *web service* funcionam dessa mesma forma, enviando um JSON do tipo Resposta com os dados e a identificação do que se

deseja fazer e recebendo um JSON do tipo Resposta com o *status* e os dados do processamento, caso existam.

Na Listagem 40 está o código responsável por desenhar a tabela com as rotas na tela.

Listagem 40 – Desenho da tabela de rotas na tela

```
<p-growl [(value)]="msgs"></p-growl>
<div class="container">
        <p-table [value]="rotas">
                <ng-template pTemplate="header">
                <tr>
                    <th>Nome da Rota</th>
                    <th>Tipo Transporte</th>
                    <th>Distância(Mts.)</th>
                    <th>Ver / Baixar / Excluir</th>
                </tr>
                </ng-template>
                <ng-template pTemplate="body" let-rota>
                <tr>
                    <td>{{rota.nomeRota}}</td>
                    <td>{{rota.tipoTransp}}</td>
                    <td>{{rota.distancia | number:'.0-0'}}</td>
                    <td style="width: 1px">
                        <button class="btn btn-primary"
alt="Visualizar Rota." (click)="visualizar(rota.idWeb)" >
                            <i class="fa fa-search"></i>
                        </button>
                        <button class="btn btn-success"
(click)=baixar(rota.idWeb) alt="Fazer download do percurso." >
                            <i class="fa fa-download"></i>
                        </button>
                        <button class="btn btn-danger"
(click)=remover(rota.idWeb)  alt="Excluir o percurso.">
                            <i class="fa fa-trash"></i>
                        </button>
                    </td>
                </tr>
                </ng-template>
            </p-table>

    </div>
```

Fonte: autoria própria.

Ao clicar na opção “Ver”, a aplicação abrirá uma nova tela contendo um mapa no qual é apresentado o traçado do percurso do usuário e, caso haja imagens, será inserido um pequeno ícone informando que ali há uma imagem que poderá ser visualizada clicando neste item, além de outras informações sobre o ponto.

Da mesma forma que nas listagens, no método ngOnInit é disparada uma consulta ao *web service*, porém desta vez são requisitados além das rotas, todos os seus dados

relacionados, como pontos e comentários existentes sobre a rota. A conexão ocorrerá da mesma forma que na listagem de rotas.

Ao receber o retorno da requisição, a aplicação *web* valida se houve algum erro de processamento e se tudo ocorreu de acordo, os dados são processados e transformados em objetos para exibição ao usuário. Na Listagem 41 está o trecho de código responsável por desenhar o mapa na tela para visualização da rota pelo usuário.

Listagem 41 – Desenhando mapa na tela

```
<agm-map [latitude]="lat" [longitude]="lng" [zoom] = "zoom"
[mapTypeId]="tipoMapa" >
      <agm-marker *ngFor="let ponto of pontosImg"
                  [latitude]="ponto.latitude"
[longitude]="ponto.longitude"
                  [agmFitBounds]="true"
                  (markerClick)="openWindow(ponto.idWeb)"
                  [iconUrl] = "{  url: markerIcon, scaledSize: {
width: 25, height: 25 } }" >

          <agm-info-window
              [isOpen]="isInfoWindowOpen(ponto.idWeb)"
              [latitude]="ponto.latitude"
              [longitude]="ponto.longitude">

              <div class="card" style="width: 30rem;">
              <img class="card-img-top"
src="{{ponto.imagemByteStr}}" alt="Card image
cap">
                <div class="card-body">
                  <h4 class="card-title"><br/>INFORMAÇÕES:<br/></h4>
<h6 class="card-text">
                    <p style="text-align: left">Latitude:
{{ponto.latitude}}</p>
                    <p style="text-align: left">Longitude:
{{ponto.longitude}}</p>
                    <p style="text-align:
left">Descrição:<br/>{{ponto.descLocalizacao}}</p>
                    <p style="text-align: left">Velocidade:
{{ponto.velocidade}} Km/h</p>
                    <p style="text-align: left">Data/Hora:
{{ponto.dataHora}}</p>
                    <a
href="https://www.google.com.br/maps/dir///@{{ponto.latitude}},{{pon
to.longitude}},1021m/data=!3m1!1e3" target="_blank" class="card-link
blank">Como Chegar</a>
                  </h6>
                </div>
              </div>
          </agm-info-window>
      </agm-marker>
```

```
      <agm-polyline  [strokeColor]="'red'">
        <ng-container *ngFor="let ponto of pontos">
          <agm-polyline-point [latitude]="ponto.latitude"
[longitude]="ponto.longitude">
          </agm-polyline-point>
        </ng-container>
      </agm-polyline>
  </agm-map>
```

Fonte: autoria própria.

Para que o mapa seja exibido também é necessária uma configuração no módulo principal da aplicação. Essa configuração e descrita na Listagem 42 e trata-se da adição da chave de liberação do projeto para uso da API de mapas da Google.

Listagem 42 – Inclusão da API Key de liberação de uso de mapas na aplicação

```
imports: [
    AgmCoreModule.forRoot({
         apiKey: 'AIzaSyCM9MWBUGkFRvw2JpIZQt9CNWkm4QZD8RX'
    })
  ],
```

Fonte: autoria própria.

Abaixo do mapa está disponível também um campo no qual os usuários podem adicionar comentário a rota. Ao clicar em salvar, uma requisição também é disparada ao *web service* contendo o comentário a ser salvo e a rota a qual esse comentário estará vinculado, além do usuário que está comentando e o identificador de operação, da mesma forma ocorre para a exclusão de comentários.

No menu superior da aplicação além da opção de listagem de rotas próprias e de outros usuários, há a opção de *logoff* a qual encerra a sessão do usuário no navegador e elimina o *token* de conexão com o *web service*, redirecionado a aplicação para a página de *login* novamente.

4 CONCLUSÃO

Com o desenvolvimento do aplicativo, resultado da realização deste trabalho, foi possível utilizar vários recursos disponíveis na plataforma *mobile*, como: sensores de acelerômetro e giroscópio que são muito utilizados na área de jogos, *players* de mídias, sensores magnéticos que são utilizados na construção de bússolas, mapas e jogos, dentre outras aplicabilidades. Também foi possível conhecer novas ferramentas *web*, como a disponibilizada pelo Angular para desenhar mapas em tela, ou também o interceptor por meio do qual se pode interceptar qualquer requisição de entrada ou saída adicionando tratativas ou informações às requisições, também foi possível entender melhor como funcionam os métodos de acesso a dados do Spring, conhecer as facilidades que as novas versões do Spring trouxeram e conhecer sua API voltada para aplicações *mobile*.

Sobre a tecnologia de mapas utilizada na aplicação, o sensor utilizado foi o de GPS. Esse sensor pode ter várias outras áreas de aplicação, como localizadores de dispositivos perdidos, rastreamento de pessoas e veículos, captura de áreas e locais. Utilizado para finalidades como direcionamento de propaganda, registro de localização do cliente em compras eletrônicas e, até mesmo, na construção de um sistema GPS. Esse sistema é semelhante ao dos dispositivos GPS disponíveis para cálculos de rotas e auxílio de deslocamento. Porém, com a vantagem de ser uma ferramenta disponível em um celular, não necessitando de um dispositivo específico que desenvolve a função única de roteamento de GPS para deslocamento e orientação.

A aplicação *web* vem com o intuito de disponibilizar aos demais usuários, experiências vividas pelos usuários em visitas a lugares diferentes e que possam, de alguma forma, contribuir na decisão de outros usuários a conhecer ou evitar determinados lugares a partir das impressões repassadas, por outros usuários que já frequentaram o local ou por imagens compartilhadas.

Em projetos para dispositivos móveis, um dos maiores desafios está no desenvolvimento do leiaute das telas de forma que sejam aplicáveis as mais variadas dimensões e resoluções de dispositivos (computadores, *notebooks*, *tablets*, *smartphones* e outros) comercializadas.

Outro grande desafio está voltado para o desenvolvimento de lógicas de aplicação de forma que o dispositivo, ao executar qualquer funcionalidade, não trave ou até aborte a aplicação. Com a adição de comunicação com serviços *web*, um novo desafio surge, a segurança dos dados transmitidos, a garantia de que o usuário que diz ter enviado a requisição

é mesmo quem a enviou, ou então, a segurança em garantir que somente determinadas aplicações consigam acessar determinadas funcionalidades presentes em um serviço *web*.

Por meio do desenvolvimento deste projeto foi possível adquirir aprendizado no uso de mapas em aplicações *mobile*. Além disso, adquiriu-se conhecimento no uso de serviços e rotinas paralelas na plataforma *mobile*. Na parte *web*, especialmente para o autor deste trabalho, por trabalhar profissionalmente apenas com aplicações *desktop*, é um mundo totalmente novo, com um leque gigantesco de opções e fantástico em termos de recursos, sejam eles gráficos ou de aplicação. São tantas possibilidades que, na maioria das vezes, acabam por confundir ou dificultar na opção de escolha de qual tecnologia utilizar. Esse é um aprendizado significativo na construção de componentes de apresentação de dados em tela de forma personalizada, possibilitando aplicativos com recursos de telas específicos e direcionados a um público alvo específico. Buscando, assim, facilidade e segurança na utilização da aplicação, o que agrega valor, tanto para aplicação como para o conhecimento do desenvolvedor.

Com o crescente uso de dispositivos móveis e do surgimento de aplicações *web* adaptáveis, até mesmo a dispositivos móveis, aliado à praticidade de seu uso e com a disponibilidade dos mais variados tipos de sensores integrados, o mercado de aplicações *mobile* é visivelmente crescente e promissor.

Em termos de funcionalidades, o aplicativo desenvolvido atende aos objetivos propostos. Como implementações futuras, melhorias em funcionalidades existentes, como na segurança de comunicação *mobile* com o *web service* bem como implementações de novas funcionalidades como o caso de *push* de notificações no APP quando da adição de novos comentários ou a possibilidade de inclusão de comentários pelo aplicativo *mobile*. Também a adição de classificações de áreas de interesses nas rotas, para que possam ser desenvolvidos novos filtros nas listagens de rotas, possibilitando que usuários possam filtrar suas áreas preferências ou, até mesmo, filtragem por proximidade do local onde o usuário está. Assim, as limitações e a necessidade de acesso em duas aplicações distintas para visualização de determinadas opções deixaram de existir.

Ainda em melhorias futuras, pretende-se modificar o modo de persistência de imagens capturadas, que hoje são persistidas em base de dados, o que, com o aumento do uso ira gerar um grande volume em espaço de armazenamento além de, na comunicação JSON gerar um arquivo gigantesco tornando a comunicação lenta e mais suscetível a falhas. Outra melhoria planejada é a criação de um cadastro do tipo de transporte utilizado para percorrer a rota, dessa forma, ao iniciar uma rota o usuário não precisara digitar o tipo de transporte.

Pensando também na privacidade dos usuários da aplicação, esta prevista a criação de uma ‘rede de amigos’ onde, ao compartilhar uma rota, o usuário poderá selecionar quem poderá vê-la, seus amigos somente ou todos os usuários da aplicação.

REFERÊNCIAS

ANATEL. **Estatísticas de celulares no Brasil**. Disponível em: <https://www.teleco.com.br/ncel.asp>. Acesso em: 11 nov. 2019.

CNOVA MARKTPLACE. **Crescimento do mercado Mobile no Brasil e no mundo**. 2015. Disponível em: <http://marketplace.br.cnova.com/artigo/crescimento-do-mercado-mobile-no-brasil-e-no-mundo/>. Acesso em: 20 ago. 2019.

EICHLER, Gerald; LÜKE, Karl-Heinz.; REUFENHEUSER, Britta. **Context information as enhancement for mobile solutions and services**. 13th International Conference on Intelligence in Next Generation Networks, Bordeaux, v. 1, n. 1, p.1-5, out. 2009.

MACHADO, Cleverson. **Aplicativo Android para mapeamento de rotas percorridas**. 2017. Monografia (graduação em tecnologia) – Câmpus Pato Branco, Universidade Tecnológica Federal do Paraná, Paraná, 2017.

MOBILE TIME. **1,5 bilhão de smartphones vendidos no mundo em 2016**. Em <http://www.mobiletime.com.br/16/02/2017/1-5-bilhao-de-smartphones-vendidos-no-mundo-em-2016/466463/news.aspx>. Acesso em: 12 ago. 2019.

MORIMOTO, Carlos E**. Receptores GPS e triangulação de sinal.** São Paulo, nov. 2010. Seção Dicas. Disponível em: <http://www.hardware.com.br/dicas/gps-triangulacao.html>. Acesso em: 12 ago. 2019.

PAVLIĆ, Daniel; PAVLIĆ, Mile; JOVANOVIĆ, Vladan. **Future of internet technologies**. In: 35th International Convention of Information Communication Technology, Electronics and Microelectronics (MIPRO 2012), mar 2016, p. 1377-1371.

PRIMEFACES. Disponível em < https://www.primefaces.org/primeng/#/> Acesso em: 11 jan. 2019.

TENG, Chia-Chi; HELPS, Richard. **Mobile application development**: essential new directions for IT. In: 7th International Conference on Information Technology: New Generations (ITNG), April 2010, *p*. 471-475.