UNIVERSIDADE DO VALE DO TAQUARI – UNIVATES
CURSO DE ENGENHARIA DA COMPUTAÇÃO

FINDED: APLICAÇÃO MOBILE PARA RASTREAMENTO GEOGRÁFICO E COMUNICAÇÃO VIA CHAT ENTRE MEMBROS DE UM GRUPO DE INTERESSE

Matheus Werlang Frey

Lajeado, novembro de 2020

Matheus Werlang Frey

FINDED: APLICAÇÃO MOBILE PARA RASTREAMENTO GEOGRÁFICO E COMUNICAÇÃO VIA CHAT ENTRE MEMBROS DE UM GRUPO DE INTERESSE

Trabalho de conclusão de curso apresentado no Centro de Ciências Exatas e Tecnológicas, da Universidade do Vale do Taquari – Univates, como parte dos requisitos para obtenção do título de Bacharel em Engenharia da Computação.

Orientador: Prof. Dr. Alexandre Stürmer Wolf

Lajeado, novembro de 2020

AGRADECIMENTOS

Agradeço a todos professores pelos ensinamentos transmitidos durante os anos de graduação, principalmente ao Dr. Alexandre Stürmer Wolf que, além de tantas disciplinas ministradas, também me orientou durante o desenvolvimento deste trabalho.

Agradeço também aos colegas pelo conhecimento e caronas compartilhadas durante as memoráveis viagens de Linha Cecília até Lajeado. Obrigado também aos voluntários que participaram dos testes do aplicativo Finded e demais pessoas que colaboraram de alguma forma com essa jornada.

Agradeço sobretudo a família. Minha namorada Ana, uma mulher inspiradora que apoiou cada etapa desta trajetória. Meus pais, Alceu e Elisabet, e meu irmão Leandro, fundamentais em todos os aspectos da minha vida.

Um agradecimento especial a minha mãe, a quem eu dedico este trabalho. Obrigado mãe, pelas passagens, marmitas, incentivos e tantos outros esforços que palavras não podem expressar.

RESUMO

Uma das dificuldades do cotidiano é ter acesso de forma rápida e eficaz sobre informações de localização de outras pessoas, como amigos, familiares ou colegas. Tentar descrever este tipo de informação por mensagem de texto ou voz e garantir que os receptores as compreendam pode levar bastante tempo, além de mostrar-se uma prática imprecisa na maioria das vezes. Buscando auxiliar pessoas a agilizarem tais tarefas, foi desenvolvido um aplicativo *mobile* para a plataforma Android que explora recursos de localização e permite conectar os usuários através de grupos, possibilitando que a geolocalização dos membros seja projetada em tempo real por meio de um mapa virtual. Da mesma forma, uma opção de *chat online* para troca de mensagens de texto foi implementada, buscando facilitar a comunicação entre os usuários. Para compreender o funcionamento do aplicativo desenvolvido, o presente trabalho apresenta um estudo sobre técnicas de localização, plataforma Android e tecnologias para dispositivos móveis, bem como uma pesquisa sobre aplicativos de geolocalização existentes no mercado e trabalhos relacionados ao tema. Também é apresentado o processo de elaboração, implementação e validação do aplicativo, que comprova a capacidade da solução criada em aproximar usuários e agilizar tarefas corriqueiras relacionadas a localização.

Palavras-chave: GPS. Mobile. Android. Firebase.

ABSTRACT

One of the daily difficulties is to have quick and effective access to other people's location information, such as friends, family, or colleagues. Trying to comment on this type of information by text or voice message and ensuring that recipients as they understand can take a long time, in addition to proving to be an imprecise practice most of the time. In search of people to help speed up these tasks, a mobile application for an Android platform was developed that explores location features and allows users to be connected through groups, allowing the location of members to be projected in real time through a virtual map. Likewise, an online chat option for exchanging text messages was implemented, seeking to facilitate communication between users. To understand the functioning of the developed application, the present work presents a study on localization techniques, Android platform and technologies for mobile devices, as well as a research on existing localization applications on the market and works related to the theme. Also presented is the process of elaborating, implementing, and validating the application, which proves the ability of the solution created to bring users closer together and streamline routine tasks related to localization.

Keywords: GPS. Mobile. Android. Firebase.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
AGPS	Assisted Global Position System
AOT	Ahead of Time
AP	Access Point
API	Application Programming Interface
ART	Android Runtime
AS	Anti-spoofing
BaaS	Backend as a Service
BO	Boletim de ocorrência
BPMN	Business Process Model and Notation
CAP	Consistency, Availability e Partition tolerence
ERP	Enterprise Resource Plann
GIS	Geographic Information System
GLONASS	Global Navigation Satellite System
GNSS	Global Navigation Satellite System
GPS	Global Position System
HTTP	Hypertext Transfer Protocol
IBGE	Instituto Brasileiro de Geografia e Estatística
IDE	Integrated Development Environment
IHC	Interação Humano Computador
IMEI	Mobile Equipment Identify
JIT	Just in Time
JSON	JavaScript Object Notation
LTE	Long Term Evolution
MV	Máquina virtual

NoSQL	Not only Structured Query Language
REST	Representational State Transfer
SDK	Software Development Kit
SIM	Micro Chip
SQL	Structured Query Language
TIC	Tecnologia da Informação e Comunicação
UML	Unified Modeling Language
URSS	União das Repúblicas Socialistas Soviéticas
XML	Extensible Makrup Language
Wi-Fi	Wireless Fidelity

SUMÁRIO

1 INTRODUÇÃO

Compartilhar informações sobre localização geográfica com outras pessoas, como amigos, familiares ou colegas, muitas vezes não é uma tarefa simples e rápida de realizar. Torna-se ainda mais complexa em determinadas situações, como quando as pessoas estão em constante movimento, não há conhecimento sobre o local em que encontram-se ou é necessário obter informações sobre a localização de várias pessoas ao mesmo tempo.

Esse tipo de informação é importante em muitas circunstâncias, como para prever horários de chegada, determinar a distância de certas referências, saber se já chegaram ao destino, evitar desencontros ou até mesmo pelo simples fato de saber se estão em locais seguros.

Pode-se citar como exemplo os pais cujos filhos estão se deslocando para algum evento em outra cidade ou região. Neste contexto, é importante saber se os filhos estão a caminho, se já chegaram ao local previsto ou se estão retornando. A mesma situação ocorre quando os pais desejam monitorar a localização dos seus filhos no dia a dia, principalmente tratando-se de adolescentes.

Assim como motivações pessoais, outras situações sociais ou profissionais também podem ser citadas, como uma empresa que deseja monitorar a localização de seus técnicos para melhorar a logística e atendimento, ou uma empresa que trabalha com planos de saúde e deseja rastrear pacientes, como idosos e portadores de deficiência. Do mesmo modo, um motorista de transporte coletivo que

deseja saber se os passageiros já estão na respectiva parada ou até mesmo o inverso, onde os passageiros precisam saber a localização exata do transporte.

De acordo com Schonarth (2017), o avanço tecnológico possibilitou que novas técnicas de localização fossem criadas, que puderam ser incorporadas em dispositivos móveis como *smartphones* e *tablets,* tornando-lhes os equipamentos mais utilizados para localização e orientação atualmente.

Estes dispositivos estão largamente difundidos entre a população mundial, conforme mostram dados divulgados pela GSM Association (GSMA), onde consta que aproximadamente 5,1 bilhões de pessoas em todo mundo usam algum tipo de dispositivo móvel, sendo que no Brasil este número chega a 204 milhões (AGÊNCIA BRASIL, 2019). Gartner (2020) prevê que até o final de 2020 este número sofra um crescimento de 3% em decorrência da introdução de cobertura 5G em diversos países.

Conforme uma pesquisa realizada em 2017 pelo Instituto Brasileiro de Geografia e Estatística – IBGE, constatou-se que 78,2% da população brasileira com 10 anos ou mais de idade possui aparelho celular para uso pessoal, e entre estes, em 97% dos casos o aparelho é o principal meio de acesso à *internet* (IBGE, 2017).

Essa disseminação está ligada a diversos fatores, conforme apontam alguns autores. Para Coutinho (2014), há relação com as diversas utilidades disponibilizadas, seja para auxiliar em tarefas corriqueiras, profissionais, de estudo, ou simplesmente por entretenimento.

Mota et al. (2016), acrescenta que a evolução da telefonia móvel também colabora com esse processo em virtude da mobilidade e cobertura proporcionadas. Por exemplo, é possível acessar rapidamente dados de qualquer lugar e a qualquer hora através de um dispositivo que cabe no bolso.

Para Cassoni, Sgarbi e Sakaguti (2017), a popularidade também é impulsionada pela concorrência comercial entre iPhone, lançado em 2007, e Android, lançado em 2008. No caso do Android, por ser desenvolvido em código aberto, é facilmente disseminado entre os fabricantes, possibilitando a comercialização de aparelhos mais baratos. Dados divulgados pela International

Data Corporation (IDC) em 2019 mostram que as plataformas Android e iOS dominam o mercado, com 87% e 13% das vendas, respectivamente, enquanto outros sistemas operacionais não chegam a 1% (IDC, 2019).

Tendo em vista este cenário, foi desenvolvido um aplicativo *mobile* para a plataforma Android que explora os recursos de localização *indoor* e *outdoor* dos aparelhos e permite compartilhar a posição em tempo real com outros usuários através de um mapa virtual. Para este compartilhamento adotou-se a dinâmica de grupos, possibilitando que qualquer usuário crie e participe deles, obtendo acesso a localização dos outros membros em tempo real.

Assim sendo, o presente trabalho apresenta um estudo sobre técnicas de localização, aplicativos de geolocalização existentes no mercado e trabalhos relacionados ao tema. O processo de elaboração, implementação e validação do aplicativo Finded também é apresentado, comprovando a viabilidade de aproximar usuários e agilizar tarefas corriqueiras relacionadas com localização.

1.1 Problema a Ser Resolvido e Público-alvo

Compartilhar informações sobre localização geográfica com outras pessoas nem sempre é uma tarefa simples e rápida de realizar. O presente trabalho visa agilizar este processo através de um aplicativo que disponibiliza tais informações em tempo real, destinado a todo público com acesso a *smartphones* e *tablets* da plataforma Android e que anseia ter acesso sobre a localização de outras pessoas, como familiares, amigos, colegas, entre outros.

1.2 Objetivos

Nas seções a seguir serão apresentados os objetivos gerais e específicos do presente estudo.

1.2.1 Objetivo Geral

O objetivo do presente trabalho é explorar técnicas de localização dos dispositivos móveis, buscando desenvolver uma aplicação *mobile* para a plataforma Android que permita conectar pessoas através de grupos de interesse, possibilitando que os membros compartilhem suas localizações em tempo real uns com os outros, além de trocarem mensagens de texto de maneira *online*.

1.2.2 Objetivos Específicos

- Analisar aplicativos de geolocalização existentes no mercado e trabalhos relacionados ao tema;
- Definir as ferramentas e tecnologias para o desenvolvimento do aplicativo proposto;
- Explorar as capacidades do Firebase;
- Desenvolver aplicativo que explore técnicas de localização da plataforma Android, possibilite compartilhar tais informações em tempo real e trocar mensagens de texto *online* com membros do grupo;
- Validar o experimento com usuários e avaliar os resultados obtidos;
- Disponibilizar o estudo realizado para auxiliar em trabalhos futuros relacionados ao tema.

1.3 Estrutura do Trabalho

O presente trabalho está dividido em 8 capítulos, onde o primeiro traz uma contextualização sobre dispositivos móveis no cenário atual e o segundo apresenta a fundamentação teórica necessária para compreender o trabalho desenvolvido. O capítulo três apresenta um estudo sobre aplicativos de geolocalização existentes no

mercado e trabalhos relacionados ao tema, enquanto o capítulo quatro descreve a metodologia utilizada para realização do estudo e as tecnologias empregadas no desenvolvimento. O capítulo cinco traz detalhes sobre a elaboração do aplicativo, como arquitetura, estruturação do banco de dados, implementação e testes. O capítulo seis apresenta os testes de validação realizados com usuários voluntários e o capítulo sete as melhorias futuras que podem ser realizadas. Por fim, o capítulo oito traz a conclusão sobre o trabalho realizado.

2 REFERENCIAL TEÓRICO

Este capítulo traz a fundamentação teórica necessária para compreender os principais conceitos do trabalho elaborado, como técnicas de localização, arquitetura da plataforma Android, mobilidade e integração dos dados com a nuvem.

2.1 Localização e mobilidade

Localização é um termo genérico para indicar a posição em que um ponto se encontra no espaço. Este termo é normalmente utilizado na área de geografia e através das coordenadas geográficas e coordenadas cartográficas é possível determinar a localização destes pontos no globo terrestre (SCHONARTH, 2017).

As coordenadas geográficas consistem em linhas imaginárias que atravessam o globo terrestre vertical e horizontalmente. A seção 2.2.1.1 do presente trabalho descreve como a localização é obtida através da interseção destas linhas.

Já as coordenadas cartográficas, segundo Cerqueira (2017), permitem reproduzir a dimensão esférica da terra em um mapa, onde uma rede de paralelos e meridianos propicia que os mapas sejam projetados. As coordenadas cartográficas, assim como as geográficas, também possibilitam localizar pontos no globo terrestre.

Schonarth (2017) explica que para obter a localização geográfica de um ponto é possível utilizar maneiras arcaicas, como bússolas e pontos cardeais, ou então equipamentos mais modernos, como os dispositivos móveis.

Os sistemas e técnicas de localização existentes nos dispositivos móveis serão abordadas nas seções a seguir, onde é possível perceber que alguns deles são mais eficientes em ambientes *outdoor* e outros em ambientes *indoor.*

2.2 Sistemas e técnicas de localização

As seções a seguir descrevem sobre as principais tecnologias e técnicas de localização existentes nos dispositivos móveis, responsáveis por determinar a posição geográfica do dispositivo. Algumas delas são baseadas em satélites e outras usam meios alternativos, como redes de telefonia e sinais de rádio frequência.

2.2.1 Localização por satélite

Os dispositivos móveis integram com diversos sistemas de localização por satélite, dos quais destacam-se o GPS, GLONASS e Galileo, que serão apresentados nas próximas subseções.

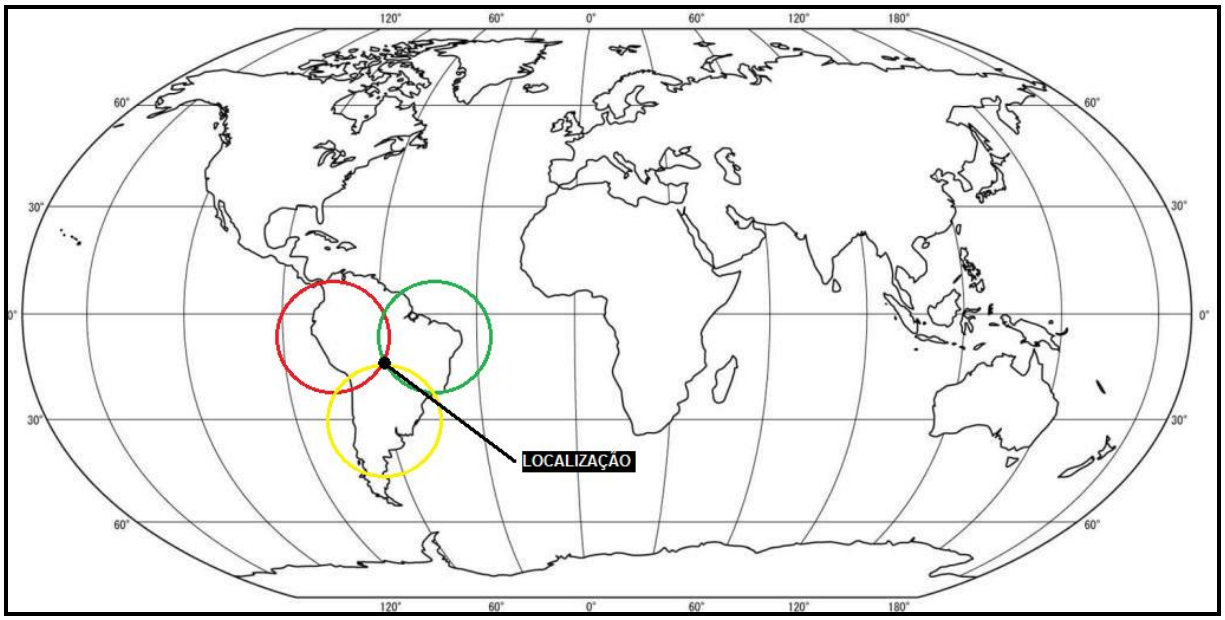
2.2.1.1 GPS

O Sistema de Posicionamento Global (GPS) foi criado pelo Departamento de Defesa dos Estados Unidos com o intuito de facilitar o deslocamento das tropas militares terrestres, aéreas e marítimas, além de auxiliar na localização de tropas inimigas. Seu funcionamento consiste em um sistema de rádio navegação programado para fornecer as coordenadas bidimensionais e tridimensionais de pontos do terreno, bem como sua velocidade e direção. Isso acontece através da determinação da distância entre um ponto chamado receptor e os pontos de referência, que são os satélites (MONICO, 2000).

Para obter a localização bidimensional é necessário que o receptor receba sinais de 3 referências, então através da interseção entre as circunferências de cada

referência, onde o raio é a distância entre o receptor e a referência, é possível determinar a posição geográfica, assim como mostra a Figura 1. Uma quarta referência permite obter as coordenadas tridimensionais, além de melhorar a precisão e o sincronismo entre os relógios (ALVES, 2006).

Figura 1 – Determinação da localização bidimensional

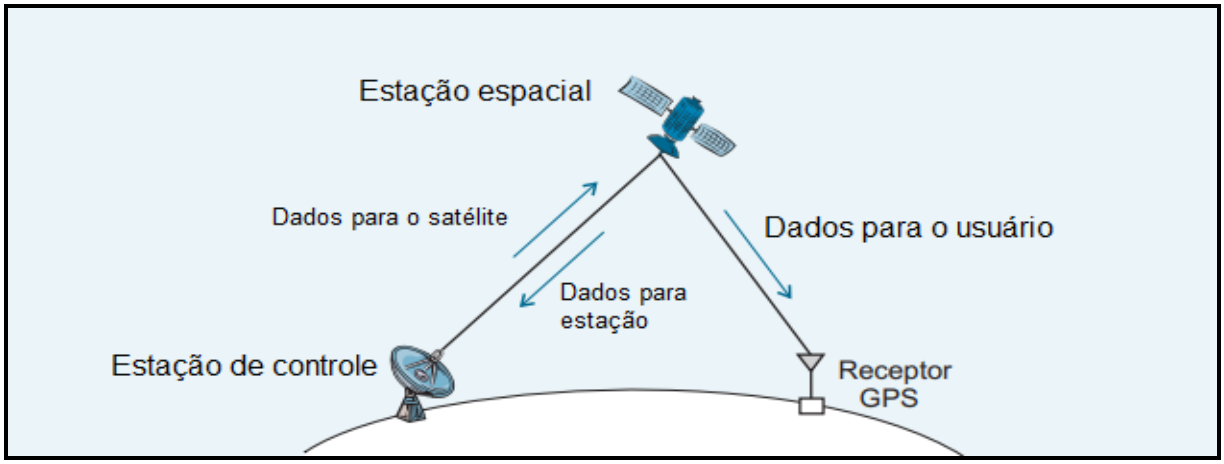


Fonte: Adaptado pelo autor com base em Alves (2006).

De acordo com NASA (2019), existem 24 satélites operacionais posicionados em torno do globo terrestre, sendo que pelo menos 4 deles podem ser vistos de qualquer lugar do planeta. Equipados com relógios atômicos, cada satélite transmite sua posição e hora em intervalos regulares, 24 horas por dia e 7 dias por semana, independente das condições climáticas.

Alves (2006) explica que o sistema GPS pode ser dividido em três segmentos conforme ilustra a Figura 2. O primeiro deles é o segmento espacial, constituído pelos satélites distribuídos em torno do globo terrestre, a uma altura aproximada de 20.2 km. O segundo é o segmento de controle, composto por estações terrestres sob controle do Departamento de Defesa Americano, cujo objetivo é monitorar, corrigir e garantir o funcionamento do sistema. Por último, o segmento de usuários, que são os receptores utilizados para aplicação do sistema.

Figura 2 – Segmentos do sistema GPS



Fonte: Adaptado pelo autor com base em Alves (2006).

Polon (2019) explica que para expressar os pontos de localização são utilizadas as coordenadas geográficas, que são obtidas através da interseção dos paralelos. Conforme Dilião (2014), as coordenadas geográficas podem ser representadas pela latitude, longitude e altitude de um determinado ponto e são calculadas da seguinte forma:

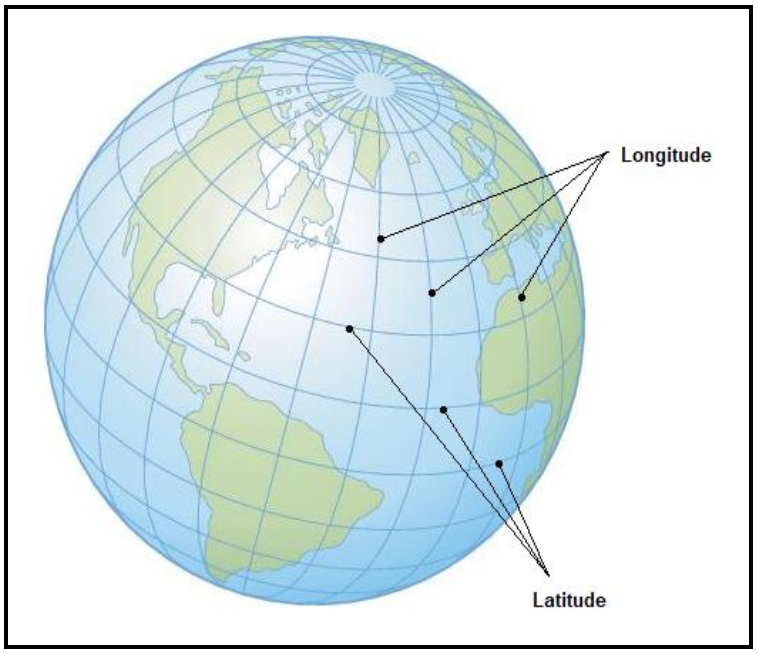
- Latitude: distância ao equador medida ao longo do meridiano de Greenwich. Essa distância é medida em graus, podendo ser de 0º a 90º para o Norte ou para o Sul. Caso as medidas sejam formadas por frações, são utilizados os minutos e segundos, que variam de 0 a 59. Valores negativos representam pontos localizados ao hemisfério sul;

- Longitude: distância ao meridiano de Greenwich medida ao longo da Linha do equador. Essa distância também é medida em graus, mas pode variar de 0º a 180º para Leste ou Oeste. Da mesma forma que a latitude, caso as medidas sejam formadas por frações, também são utilizados minutos e segundos. Valores negativos representam pontos localizados na parte ocidental do planeta;

- Altitude: o formato da Terra é esférico com achatamento nos polos, logo, denomina-se como esfera geoide. Dado que o raio desta esfera é de 6.378 Km, pode-se dizer que a altitude significa a distância vertical do centro da terra até a superfície da esfera geoide.

A Figura 3 mostra a representação dos paralelos, que são linhas traçadas paralelamente à Linha do equador para determinar a latitude, e os meridianos, que são linhas traçadas paralelamente em relação ao Meridiano de Greenwich para determinar a longitude.

Figura 3 – Linhas imaginárias para obtenção da latitude e longitude



Fonte: Adaptado pelo autor com base em Dilião (2014).

Como as coordenadas são representadas em graus, minutos e segundos, é necessário fazer uma conversão para valores decimais afim de utilizar em sistemas computacionais. Sabendo que 1 grau equivale a 60 minutos, a Equação 1 determina as coordenadas em graus decimais (MACHADO, 2015):

$$\text{Graus decimais} = -\left(graus + \left(\frac{minutos'}{60} \right) + \left(\frac{segundos'}{3600} \right) \right) \quad (1)$$

Exemplo para converter Latitude -26º 04’ 51’’ e Longitude -53º 03’ 18’’:

Latitude = - (26 + (04’ / 60) + (51’ / 3600)) = -26.080833

Longitude = - (53 + (03’ / 60) + (18’ / 3600)) = -53.055000

2.2.1.2 GLONASS

Assim como o GPS, o Global Navigation Satellite System (GLONASS) também foi criado com fins militares pela extinta União Soviética (URSS – atual Rússia). O desenvolvimento do sistema começou em 1976, mas só tornou-se operacional em 2011, quando a quantidade mínima de satélites necessários para prover cobertura global foi colocada em órbita (VAZ; PISSARDINI; DA FONSECA JUNIOR, 2013).

O GPS e o GLONASS possuem características semelhantes, tanto no aspecto orbital quanto no aspecto terrestre, e por este motivo os dois sistemas são utilizados com os mesmos fins. Uma das diferenças entre os sistemas é que o GPS utiliza uma técnica de segurança para impedir que usuários não autorizados obtenham determinadas coordenadas, enquanto o GLONASS não as utiliza. Esta técnica é chamada de *Anti-spoofing (AS),* e consiste em criptografar um código X combinado ao código secreto Y e resultando no código Z, acessível apenas por militares norte-americanos e usuários autorizados. Historicamente, o GPS foi mais difundido entre a comunidade devido a disponibilidade das informações, enquanto o GLONASS só tornou-se acessível após a dissolução da URSS (DO LAGO; FERREIRA; KRUEGER, 2002).

2.2.1.3 Galileo

Segundo Júnior, Alves e Gouveia (2016), com o surgimento do GPS na década de setenta, a partir da década de oitenta os países da Europa uniram-se para desenvolver um sistema de posicionamento independente, o Galileo, que em 2003 foi concluído, tendo uma arquitetura muito similar aos sistemas Americano e Russo. Conforme Menezes (2019), em 2019 o sistema Galileo encontrava-se com 17 satélites operacionais, entretanto existe a previsão de completar a constelação em 2020, com 30 satélites operacionais em órbita.

2.2.1.4 Comparação entre os sistemas de posicionamento por satélite

O sistema de GPS é constituído por 24 satélites operacionais. Eles estão posicionados em 6 planos orbitais com inclinação de 55º em relação a Linha do equador e altura aproximada de 20.200 Km, levando aproximadamente 12 horas para completar a órbita terrestre. Já o GLONASS possui 24 satélites operacionais, distribuídos em 3 planos orbitais a uma altura aproximada de 19.100 Km. Cada satélite possui inclinação de 64,8º em relação a Linha do equador e leva cerca de 11 horas e 15 minutos para completar a órbita terrestre. Por último, o sistema Galileo conta com 17 satélites operacionais, distribuídos em 3 planos orbitais, com inclinação de 56º em relação a Linha do equador, a uma altura de 23.222 Km e leva cerca de 14 horas para completar a órbita terrestre (MENEZES, 2019).

A tabela abaixo mostra um resumo da comparação entre os sistemas abordados no parágrafo anterior, considerando a quantidade de satélites existentes na data em que o presente trabalho foi desenvolvido.

Tabela 1 – Resumo das características do GPS, Glonass e Galileo

	GPS	**GLONASS**	**Galileo**
Satélites operacionais	24	24	17
Inclinação de órbita	55º	64,8 º	56 º
Planos orbitais	6	3	3
Período orbital	12h	11h 15min	14h
Altitude	20.200 Km	19.100 Km	23.222 Km

Fonte: Adaptado pelo autor com base em Menezes (2019).

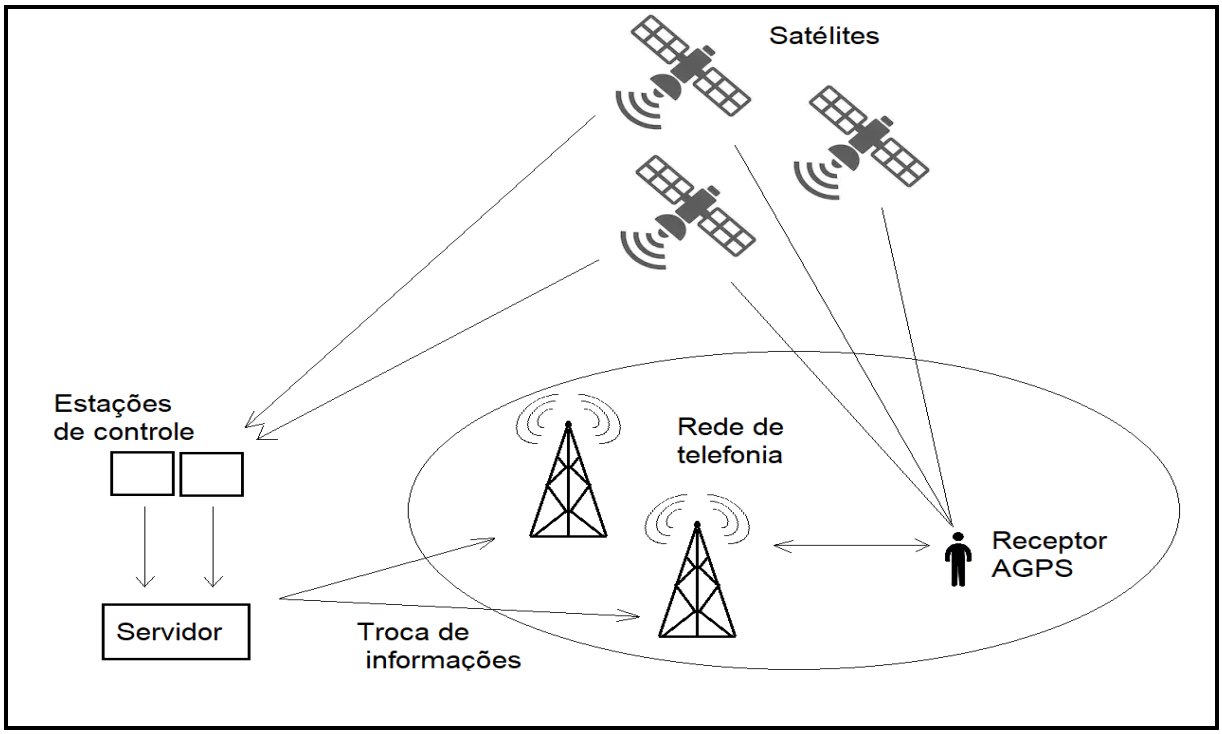
Menezes (2019) acrescenta que os sistemas GPS, GLONASS e GALILEO fazem parte do Global Navigation Satellite System – GNSS e por isso há interoperabilidade entre eles. Ou seja, é possível trabalhar combinadamente uns com os outros. Para Do Lago, Ferreira e Krueger (2002), o uso combinado resulta em algumas melhorias na precisão da localização e na área de cobertura abrangida, visto que há mais satélites disponíveis em órbita do que cada sistema teria individualmente.

2.2.2 A-GPS

Para Zandbergen e Berbeau (2011), o Sistema de Posicionamento Global Assistido, ou *Assisted* GPS (A-GPS), consiste em uma técnica de localização que faz o uso combinado de dados recebidos dos satélites GPS com dados recebidos das torres de telefonia móvel.

A Figura 4 ilustra uma visão geral do seu funcionamento, onde os dados de suporte dos satélites são capturados pelas redes de telefonia e depois de serem processados, são enviados para os receptores através de conexões Wi-Fi, 3G, 4G e afins. Estes dados de suporte auxiliam os receptores a calcular as coordenadas assim que eles recebem as demais informações dos satélites (LIMA, 2018).

Figura 4 – Visão geral do funcionamento do A-GPS



Fonte: Adaptado pelo autor com base em Lima (2018).

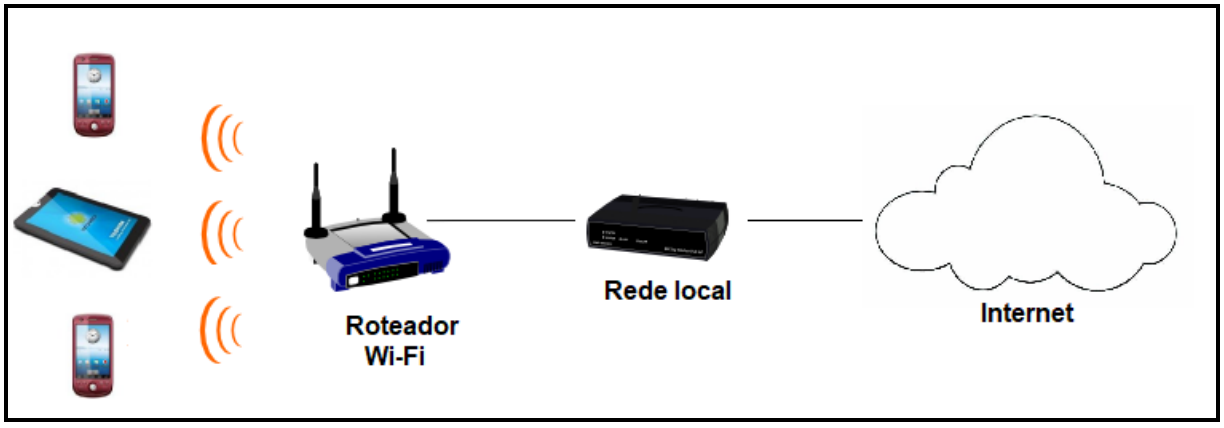
Conforme Lima (2018), essa técnica apresenta algumas vantagens em relação ao GPS. Por exemplo, é mais rápido para calcular a posição, o consumo de bateria do aparelho é menor e a precisão em lugares fechados é mais eficaz, visto que o GPS funciona melhor em ambientes *outdoor.*

Zandbergen e Berbeau (2011) explicam que essa melhoria dá-se pelo fato de que, com o uso de A-GPS, o receptor não precisa obter todos os dados diretamente do satélite, o que leva bastante tempo se comparado com o tempo gasto usando auxílio das torres de telefonia. Lima (2018) ainda acrescenta que com o auxílio da rede de telefonia, o volume de dados transmitido é menor, visto que eles são processados antes de serem enviados, sem contar que a transmissão pela rede de celulares é mais rápida do que transmitir direto do satélite.

2.2.3 Wi-Fi

Segundo Figueiredo (2016), redes Wi-Fi são redes sem fio baseadas no padrão IEEE 802.11 que podem disponibilizar acesso à *internet* para outros dispositivos. Essas redes são bastante comuns em *shoppings*, praças, restaurantes, estabelecimentos comerciais e diversos outros. Elas contam com mecanismos de autenticação, criptografia e integridade dos dados. A Figura 5 representa a topologia de rede Wi-Fi através de um *Access Point* (AP) e dispositivos de acesso *smartphone* e *tablet.*

Figura 5 – Topologia de redes Wi-Fi



Fonte: Adaptado pelo autor com base em Figueiredo (2016).

Uma característica relevante para o desenvolvimento deste trabalho é que as redes Wi-Fi também oferecem serviços de localização. Santos (2018) afirma que esse serviço tem atraído muitos esforços tecnológicos porque é útil em ambientes

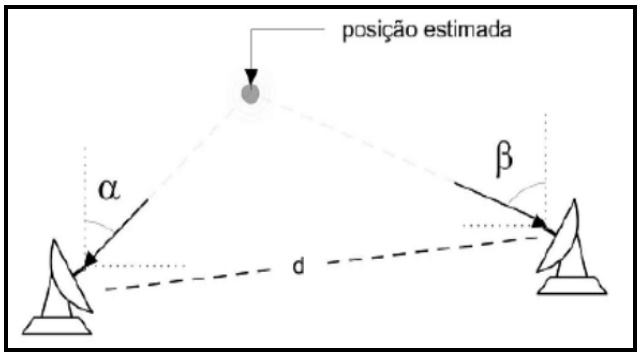
fechados, como escolas, edifícios, aeroportos e outros ambientes onde os sistemas como GPS e A-GPS não oferecem tanta precisão.

Neste caso a infraestrutura é aproveitada, onde os APs fazem o papel de referência para calcular a posição do receptor. Isso acontece porque os sinais podem ser emitidos ininterruptamente pelos APs e a intensidade deles pode ser medida pelos receptores através de uma técnica denominada *Received Signal Strength Indicator* (RSSI) (SANTOS, 2018).

Moura (2007) acrescenta que apesar deste tipo de ambiente possa sofrer com problemas de propagação do sinal, a localização acaba se tornando precisa porque geralmente as áreas de cobertura dos APs não são muito grandes.

Para determinar a localização utilizando transmissores de radiofrequência, como é o caso da Wi-Fi, Moura (2007) explica que podem ser utilizadas duas técnicas. A primeira delas consiste em calcular a posição do receptor pelo ângulo de chegada do sinal. Para isso são usadas duas referências, que calculam o ângulo do sinal que está chegando até elas e formam um triângulo com o receptor, onde o tamanho da aresta adjacente é conhecida. A Figura 6 ilustra essa técnica:

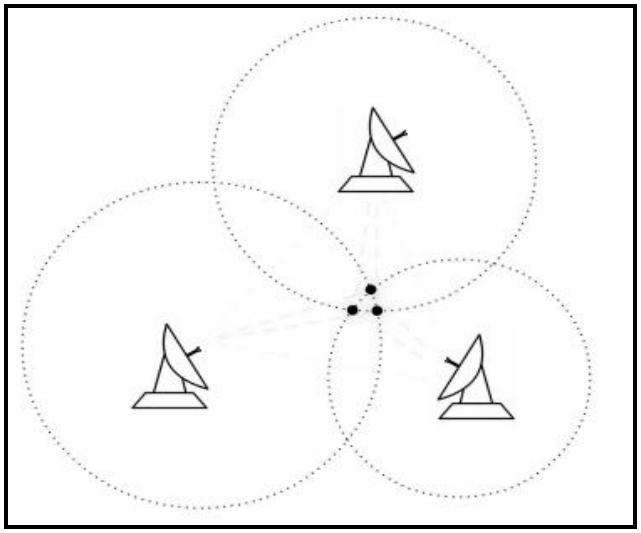
Figura 6 – Cálculo das coordenadas através do ângulo de chegada



Fonte: Adaptado pelo autor com base em Moura (2007).

A segunda delas consiste em calcular as coordenadas através da distância entre o receptor e as referências, semelhante ao GPS. Neste modelo são necessários três pontos de referência que estejam dentro do raio de alcance do receptor. Basta então traçar circunferências em torno das referências, onde o raio é a distância entre a referência e o dispositivo e, em seguida, fazer a interseção das três circunferências como mostra a Figura 7 (MOURA, 2007):

Figura 7 – Cálculo da posição através da distância do dispositivo

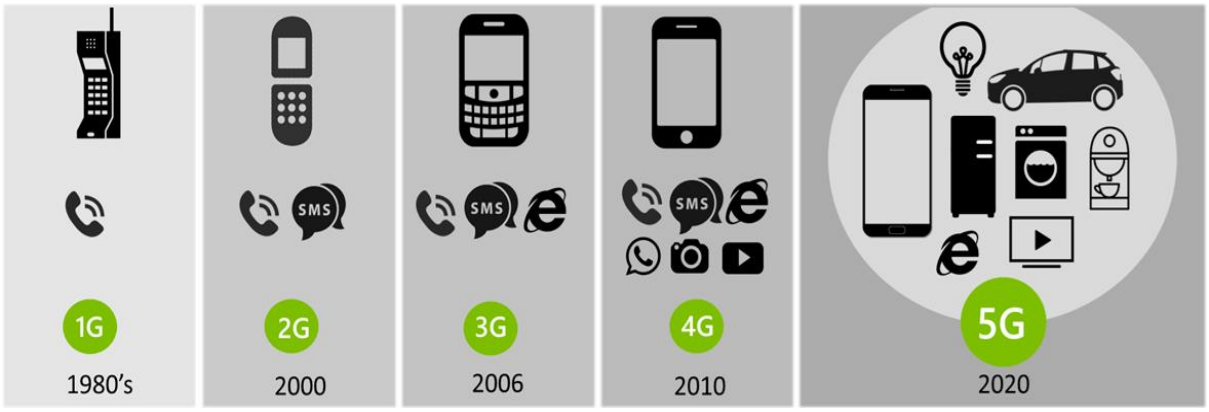


Fonte: Adaptado pelo autor com base em Moura (2007).

2.3 Serviços de telefonia

Os avanços tecnológicos na área de telefonia móvel possibilitam que dados sejam acessados de qualquer lugar, a qualquer hora e através de comunicações sem fio. Esses avanços revolucionaram o modo como as pessoas comunicam-se, principalmente pela mobilidade oferecida (RICARDO; SILVEIRA, 2016). Conforme Mota et al., 2016, na medida em que essas tecnologias foram evoluindo, os serviços e funcionalidades dos *smartphones* foram crescendo como mostra a Figura 8.

Figura 8 – Evolução das tecnologias de telefonia



Fonte: Adaptado pelo autor com base em (Silva, 2016).

A Primeira Geração (1G) surgiu na década de 80 e oferecia apenas serviços de voz. Um dos principais problemas é que não havia um padrão de comunicação entre os fabricantes Americanos, Europeus e Japoneses, fazendo com que os usuários tivessem problemas ao viajar entre estas áreas de cobertura (MOTA; et al, 2020).

A Segunda Geração (2G) surgiu na década de 2000 e foi marcada pela evolução de sistemas analógicos para sistemas digitais, dando fim aos ruídos nos serviços de voz. Além da parte técnica, também houve avanços na parte comercial, já que novos serviços foram implantados. Entre eles, a troca de dados com possibilidade de criptografia (SILVA, 2016).

A Terceira Geração (3G) surgiu como uma ascensão a tecnologia 2G, onde a principal novidade foi a oferta de banda larga sem fio. Com o aumento significativo nas taxas de transmissão de dados, outros serviços também surgiram, como acesso à *internet* e serviços de multimídia (SILVA, 2016).

A Quarta Geração (4G) foi marcada pelo surgimento da tecnologia Long Term Evolution (LTE *Advanced)*, que aumentou ainda mais as taxas de transmissão, oferecendo uma largura de banda de 100 MHz. O tempo de latência também diminuiu consideravelmente, para 10 milissegundos, fazendo com que o *downlink* possa atingir 1 Gbps e o *uplink* 0,5 Gbps (MOTA, et al; 2020).

Mota (et al; 2019), explica que os serviços de Quinta Geração (5G) já estão sendo implementados em alguns países. Entre as melhorias está a eficiência, uma vez que a velocidade de transmissão pode chegar a uma taxa de *downlink* superior 20 Gbps e *uplink* acima 10 Gbps, e a diminuição da latência de rede, que deve ser inferior a 1 milissegundo.

2.4 Plataforma Android

Android é um sistema operacional de código aberto construído com o objetivo de permitir aos desenvolvedores criarem aplicações móveis que tirem o maior proveito possível que os aparelhos *smartphone* possam oferecer (LECHETA, 2013).

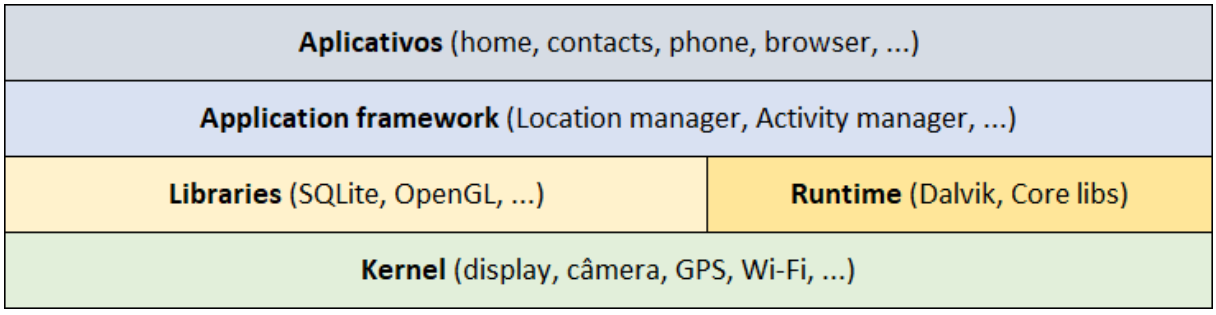
Conforme Pereira e Da Silva (2009), o desenvolvimento teve início em 2003 através da empresa Android Inc até que em 2005 o Google a comprou com o intuito de entrar no ramo de *smartphones*. Lecheta (2013) também explica que o sistema operacional Android oferece uma interface rica, integração com GPS, diversas aplicações já instaladas e um ambiente de desenvolvimento poderoso, inovador e flexível.

Para Bohrer (2011), embora a plataforma Android tenha sido desenvolvida para telefonia móvel, ela também vem sendo bastante utilizada em outros dispositivos portáteis, como *tablets*, televisões e até micro-ondas. Apesar desta variedade, o presente trabalho tem como foco o mercado de telefonia móvel e *tablets*.

2.4.1 Arquitetura da plataforma Android

A plataforma Android pode ser dividida em camadas que conectam diferentes componentes de *software*. A Figura 9 representa a arquitetura Android e as seções seguintes descrevem cada camada da arquitetura.

Figura 9 – Arquitetura plataforma Android



Fonte: Adaptado pelo autor com base em Bohrer (2011).

O *kernel* pode ser definido como um componente do sistema operacional responsável por fazer a comunicação direta com o *hardware* do equipamento. Essa camada é responsável por acessar a câmera, display, antena Wi-Fi, memória, teclado, entre outros recursos do equipamento (BOHRER, 2011).

A arquitetura da plataforma Android baseia-se no *kernel* versão 2.6 das distribuições Linux. Segundo Bohrer (2011), o motivo pelo qual essa versão foi usada como base é o gerenciamento de drivers altamente funcional, além da possibilidade de realizar o gerenciamento de memória, gerenciamento dos processos e outros aspectos comprovadamente funcionais.

Já a camada de Biblioteca possui um conjunto de bibliotecas escritas na linguagem de programação nativa, C/C++, cuja finalidade é prover acesso a componentes e serviços principais do sistema Android. Algumas das funcionalidades e recursos destas bibliotecas podem ser acessadas pelo Java Framework API (ANDROID DOCUMENTATION, 2020). Entre as bibliotecas, destacam-se:

Tabela 2 – Bibliotecas da plataforma Android

Biblioteca	*Descrição*
Surface Manager	Responsável pelo gerenciamento da exibição de diferentes telas, geradas por diferentes aplicativos, que são executados em diversos processos.
OpenGL / ES	Realiza o tratamento de imagens em 3D. Apesar de ser implementada em software, pode ser acelerada por hardware caso o aparelho possua chip para processamento de gráficos em 3D.
SGL	Realiza o tratamento de imagens 2D, as quais a maioria dos aplicativos móveis utiliza.
FreeType	Realiza a renderização de imagens do tipo *bitmap* que são utilizadas por aplicativos.
SQLite	Biblioteca que oferece suporte ao armazenamento de dados utilizando um banco de dados SQL embutido na plataforma.
Media Framework	Oferece suporte para gravação e execução de áudios e vídeos.
WebKit	Motor do navegador de *internet* utilizado pelos browsers.

Fonte: Adaptado pelo autor com base em Bohrer (2011).

O Android Runtime, por sua vez, tem como objetivo permitir que as necessidades de execução dos aplicativos sejam plenamente atendidas, dado que

os dispositivos embarcados nos quais eles são executados possuem limitação de bateria, memória RAM e CPU (BOHRER, 2011).

Para isso, Aron e Hanácek (2015) explicam que os aplicativos executados no Android utilizam uma máquina virtual (MV) denominada Dalvik. Desta forma, os programas escritos em Java são convertidos em código de máquina, chamados de *bytecode*, até que a MV Dalvik os traduza em código legível para a máquina e os execute. Esse processo é chamado de *Just In Time* (JIT) porque a tradução é feita durante a execução do aplicativo.

Para as versões 4.4 ou superiores do Android, o Google lançou a plataforma Android Runtime (ART), cujo objetivo é traduzir todo o código Java em *bytecode* antes de executá-lo. Esse processo é chamado de *Ahead Of Time* (AOT). Aron e Hanácek (2015), acrescentam que as principais vantagens da tecnologia ART em relação a MV Dalvik é a velocidade de execução maior e o consumo de bateria menor em relação ao JIT. Por outro lado, o uso de memória é mais elevado, visto que toda tradução do aplicativo precisa ser armazenada antes de rodá-lo.

Na camada Framework ficam as *Application Programming Interface* (APIs) do Android que são utilizadas pelas aplicações que executam sobre a plataforma. Pode-se citar como exemplo os gerenciadores de serviço de telefonia, localização e notificação (BOHRER, 2011). Dentre elas, destacam-se :

Tabela 3 – Frameworks da plataforma Android

Framework	*Descrição*
Location Manager	É a classe nativa do Android responsável por obter a localização geográfica do dispositivo, seja por GPS, triangulação com torres de telefonia ou redes Wi-Fi.
Notification Manager	Padroniza as notificações entre os aplicativos.
Activity Manager	Gerencia o ciclo de vida dos aplicativos e permite realizar integração entre eles.
Package Manager	Gerencia os aplicativos que estão instalados no aparelho.
Telephony Manager	Contém as APIs necessárias para utilização do dispositivo como telefone.

Content Providers	Ferramenta para que os aplicativos instalados compartilhem dados entre eles.
Resource Manager	Permite armazenar imagens, informações de layout, entre outras informações que não forem código dos aplicativos instalados.
View System	Disponibiliza acesso a botões, listas e outros componentes da interface gráfica. Também é responsável por disparar eventos e layout dos aplicativos.

Fonte: Adaptado pelo autor com base em Bohrer (2011).

Por fim, a camada de Aplicações é onde estão os aplicativos que serão utilizados pelo usuário. Entre eles o gerenciador de e-mail, calendário, navegador de *internet*, jogos, entre outros (BOHRER, 2011).

2.4.2 Android SDK

O Android Software Development Kit (SDK), é um pacote de ferramentas que auxilia os desenvolvedores a criarem aplicativos para plataforma Android. É composto por um emulador de *smartphone*, ferramentas utilitárias e uma API completa para a linguagem Java. Embora possa ser executado por linhas de comando, também é possível utilizá-lo em um ambiente de desenvolvimento integrado (IDE), com o Android Studio, Eclipse ou Netbeans (LECHETA, 2013).

2.5 Banco de dados NoSQL

Para melhor compreender o funcionamento de um sistema NoSQL, pode-se fazer uma comparação com os já consagrados sistemas SQL. Date (2014), faz a seguinte definição entre os dois sistemas:

- Relacional (SQL): Há uma relação entre os dados representada por meio de tabelas. Este modelo respeita as propriedades Atomicidade, Consistência, Isolamento, Durabilidade (ACID) cujo objetivo é garantir a integridade e o

funcionamento correto do sistema. As transações são realizadas através da linguagem *Structured Query Language* (SQL) e entre os mais comuns estão MySQL e PostgreSQL;

- Não Relacional (NoSQL): Este tipo de banco de dados não apresenta todas as características ACID. Eles se baseiam no teorema *Consistency, Availability e Partition tolerence* (CAP), portanto determinado momento só é possível garantir duas das três propriedades, entre consistência, disponibilidade e tolerância à partição. Outra característica é que não usam a linguagem SQL para realizar as transações. Também conhecidos como *Not only SQL* (NoSQL), dentre os mais conhecidos estão o Cassandra e MongoDB.

Silva (2019) afirma que os sistemas NoSQL surgiram pela necessidade de haver uma performance melhor e alta escalabilidade em comparação aos bancos relacionais. Entretanto, Saladge e Fowler (2013) acrescentam que, ao contrário do que aparenta, os bancos de dados NoSQL não surgiram com o intuito de substituir os bancos de dados relacionais, mas sim para complementá-los, direcionando seu uso em volumes de dados gigantescos e resolvendo problemas de gargalo existentes nos sistemas relacionais.

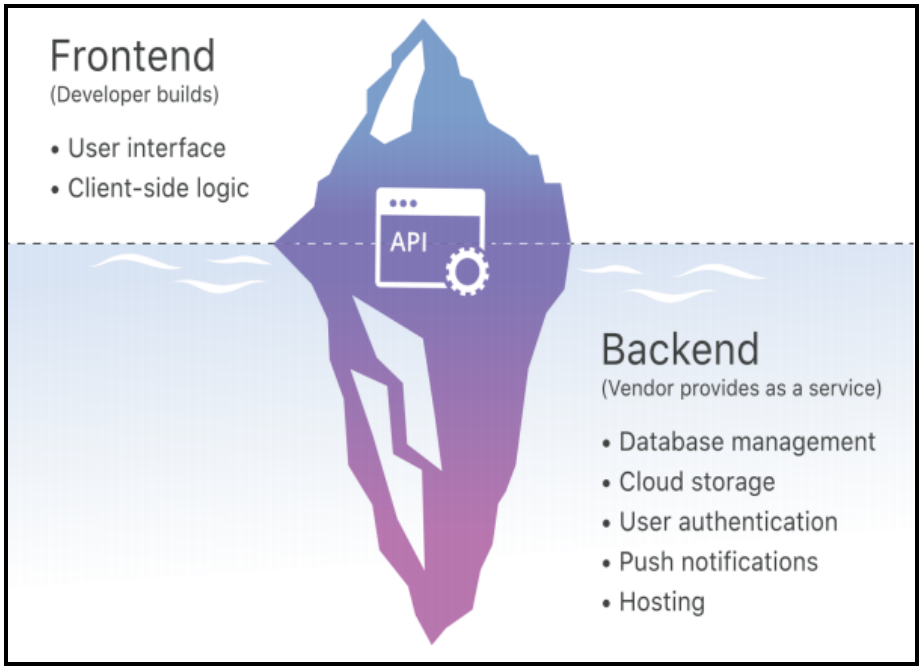
Entre as categorias de bancos de dados NoSQL estão os bancos de dados orientados a documento, que conforme Ramos (2014), são compostos por uma estrutura de dados semiestruturados e livres de *schemas.* Neste modelo, a palavra documento refere-se a um conjunto de dados compostos por pares de chave-valor, normalmente no formato XML ou JSON.

2.6 Serviços BAAS

Para Matias (2019), o *Backend-as-a-service* (BAAS) pode ser considerado como um serviço de computação na nuvem que absorve toda parte do servidor de uma aplicação *web* ou *mobile,* ou seja, toda parte do *backend.* Este tipo de serviço permite aos desenvolvedores conectar os aplicativos *mobile* aos serviços BAAS através de SDKs e APIs, auxiliando e acelerando o desenvolvimento do *frontend*, visto que toda parte de *backend* já está pronta.

A Figura 10 ilustra o tamanho que o *frontend* e *backend* possuem dentro de uma aplicação, mensurando o tempo economizado ao utilizar um serviço BAAS e as funcionalidades que ele oferece.

Figura 10 – *Frontend* e *Backend*



Fonte: Adaptado pelo autor com base em Matias (2019).

Entre as plataformas que disponibilizam o modelo BAAS destaca-se o Firebase, que foi adotado para o desenvolvimento deste trabalho para realizar a persistência dos dados locais, disponibilização de dados na nuvem, gerenciamento de seções dos usuários e envio de notificações *push* para mensagens do *chat*.

O Firebase foi lançado em 2012 pela empresa Firebase com foco no desenvolvimento de aplicativos móveis e aplicações web de menor complexidade, atuando como um *middleware* para serviços como banco de dados e autenticação, visando acelerar o desenvolvimento de aplicativos (BRAGA; DA SILVA, 2018).

Em 2014, foi adquirido pelo Google e diversos serviços foram disponibilizados na plataforma, como o Cloud Firestore, Realtime Database, Authentication, Storage, AdMob, Analytics, entre outros (TERRERI; DOS SANTOS; DA SILVA, 2018).

Lummertz (2018), destaca pontos importantes em relação ao Firebase, como a alta performance, disponibilidade e desempenho dos servidores, dado que o

Firebase usa a infraestrutura do Google e faz o dimensionamento automático para que o desenvolvedor não precise se preocupar em atender a demanda dos usuários.

Este capítulo apresentou o embasamento teórico relacionado ao tema do presente trabalho e no capítulo 3 será apresentado um estudo sobre outros trabalhos e aplicativos também relacionados ao tema.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta um estudo sobre aplicativos e trabalhos relacionados com a área de estudo do presente trabalho, onde buscou-se avaliar como foi feita a utilização das técnicas de localização, ferramentas utilizadas, funcionalidades disponíveis aos usuários e resultados obtidos. Estes trabalhos serviram como apoio para o desenvolvimento do aplicativo proposto e para elucidar a pesquisa, a última seção traz um comparativo entre as principais características de cada trabalho com as características do aplicativo proposto neste trabalho.

3.1 Zenly

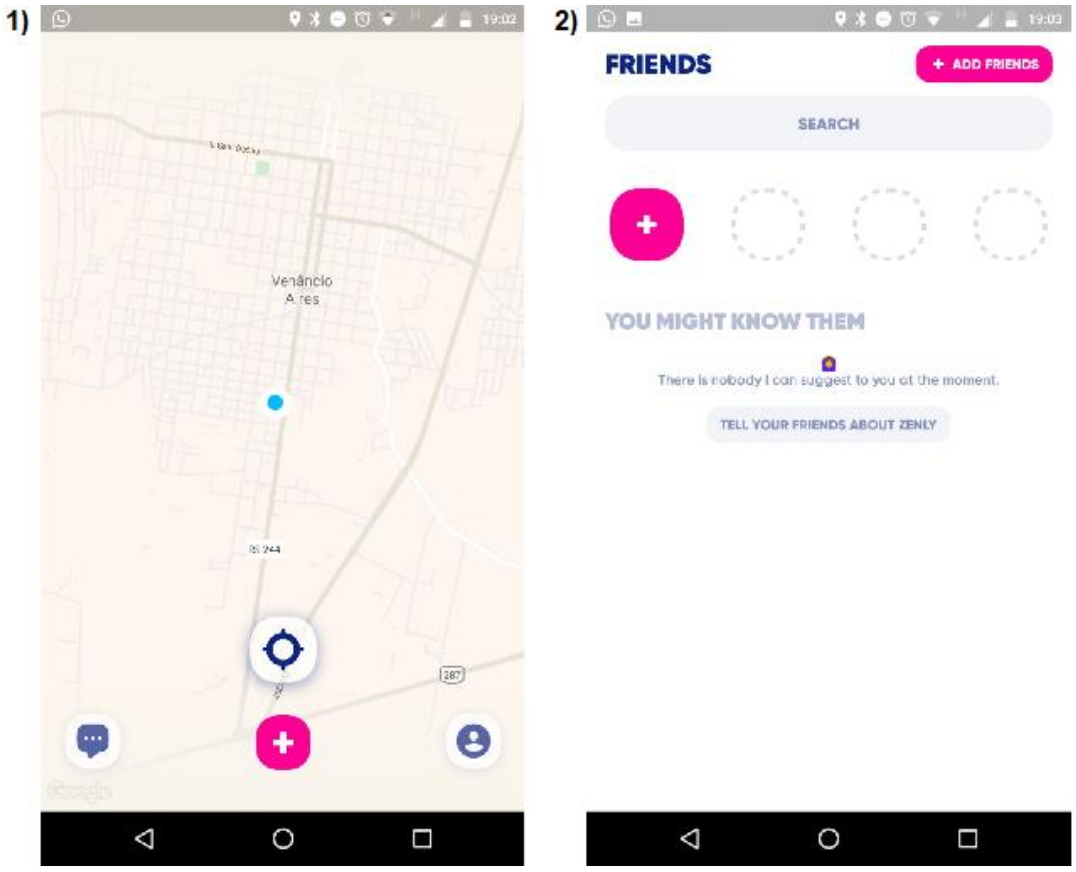
Através do seu site oficial, identificou-se que o aplicativo permite ao usuário adicionar uma rede de amigos e compartilhar a localização em tempo real com eles. Um *chat online* também é disponibilizado para que os amigos troquem mensagens individuais ou em grupos (ZENLY, 2020).

Por padrão o aplicativo sempre mostra a localização atual do dispositivo, entretanto existe uma funcionalidade denominada “Modo fantasma” que permite ao usuário interromper o compartilhamento da sua posição por questões de privacidade. Neste caso, apenas a última localização fica sendo exibida para a rede de contatos. O modo fantasma também permite que o usuário compartilhe a sua localização aproximada, sendo possível configurar um raio de precisão que varia entre 10m e 1,2km. Também é possível bloquear outros usuários para eles não

terem acesso a localização. Apesar de possibilitar que uma rede de amigos seja criada e até que mensagens em grupo sejam trocadas, o aplicativo não permite a criação de grupos.

O item 1 da Figura 11 ilustra a tela principal do aplicativo, onde é exibido o mapa geográfico com a localização atual do usuário e a localização dos amigos. Já o item 2 apresenta a lista de amigos do usuário, onde é possível fazer uma busca por número do telefone celular e por Whatsapp. Para adicionar um amigo, é necessário enviar uma solicitação de amizade e aguardá-lo aceitar.

Figura 11 – Tela principal e tela para listagem dos amigos



Fonte: Adaptado pelo autor com base em ZENLY (2020).

Informações técnicas sobre o aplicativo não são disponibilizadas pelo site oficial, como por exemplo, o banco de dados utilizado ou a ferramenta de desenvolvimento. O aplicativo conta com mais de 10 milhões de usuários e requer a versão do Android 5.0 ou superior.

3.2 Family Tracker

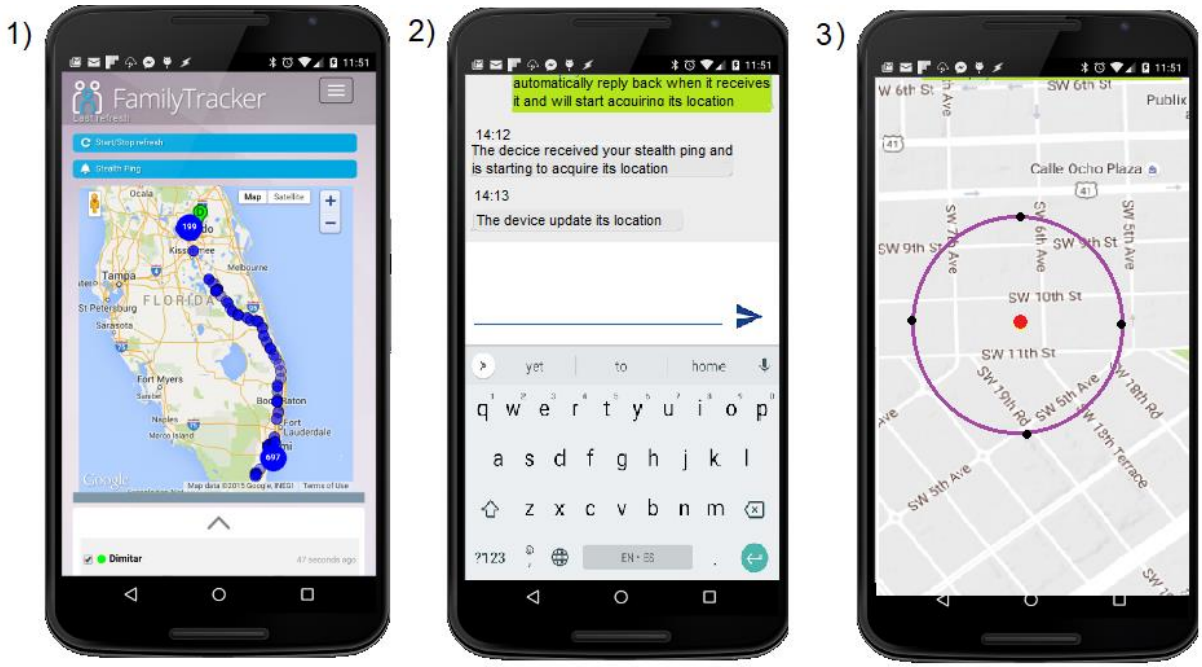
De acordo com seu site oficial, Family Tracker é um aplicativo pago de rastreamento GPS disponível para Android e iOS. Ele usa todos os métodos possíveis para localização: GPS, triangulação por torres de telefonia celular e Wi-Fi.

Com mais de 10 mil *downloads*, o aplicativo permite rastrear um número ilimitado de dispositivos e, entre os principais recursos, destacam-se:

- *Messaging:* Permite que os usuários troquem mensagens de texto privadas conforme mostra o item 1 da Figura 12;
- *Breadcrumbs*: O aplicativo armazena todos os dados e rotas do dispositivo, possibilitando que o usuário armazene estes dados ou compartilhe-os na versão paga. O item 2 da Figura 12 mostra um exemplo do percurso realizado pelo usuário;
- *Geofacing*: Consiste em enviar uma notificação quando um dispositivo estiver entrando ou saindo de uma determina área geográfica como mostra o item 3 da Figura 12. Esse recurso está disponível apenas em versões pagas e em dispositivos com iOS 5 ou superior e Android 4.4 ou superior.

O site oficial do Family Tracker também enfatiza que o aplicativo não é executado constantemente em segundo plano, resultando em um uso mais eficiente da bateria do aparelho. Além dos recursos *mobile*, o Family Tracker também pode ser acessado pela *Web* para visualizar as rotas de determinados dispositivos através de um *Access Code.*

Figura 12 – Recursos *Breadcrumbs*, *Messaging* e *Geofacing*



Fonte: Adaptado pelo autor com base em Family Tracker (2020).

3.3 Pets: Desenvolvimento de sistema de geolocalização para o monitoramento de animais de estimação

O respectivo trabalho trata sobre o desenvolvimento de uma aplicação *mobile* para monitorar a localização geográfica dos animais de estimação através de um equipamento instalado na coleira, capaz de obter e enviar as coordenadas geográficas via rede Wi-Fi e 3G.

O objetivo da aplicação é facilitar a busca dos animais pelo dono, evitando buscas exaustivas e ineficazes ou possíveis perdas traumáticas, considerando que muitas vezes os donos tratam seus animais como parte da família e cada vez mais preocupam-se com o bem estar deles.

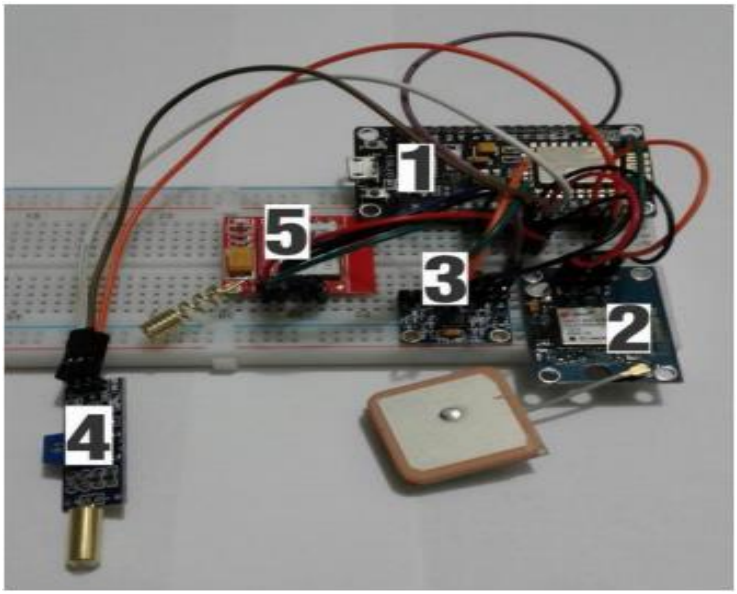
O desenvolvimento da solução é dividido em duas etapas, a primeira delas relacionada com a construção da coleira do animal e a segunda com o desenvolvimento da aplicação *mobile*.

Para construção da coleira, é utilizada a placa NodeMCU, que atua como um *shield* para programar o microcontrolador ESP-8266, que possui um módulo para conexão Wi-Fi e uma memória *flash* para armazenamento dos dados.

Também é utilizado um módulo com sensor de inclinação para identificar possíveis quedas do animal, acelerômetro com giroscópio que trata de confirmar a queda relatada pelo módulo anterior, GPS para captura das coordenadas geográficas e um módulo de conexão a redes 3G para envio de dados caso a conexão Wi-Fi não esteja disponível.

Para o funcionamento dos sensores e módulos, a plataforma Arduíno é utilizada na execução da lógica de controle dos componentes. Assim que a coleira é ligada, o módulo NodeMCU inicia a leitura dos dados e trata de fazer o envio. A Figura 13 mostra o protótipo da coleira de monitoramento, onde item 1 mostra o módulo para conexão Wi-Fi, o item 2 mostra o módulo GPS, o item 3 acelerômetro e giroscópio, o item 4 mostra o sensor de inclinação e o item 5 mostra o módulo de conexão a redes 3G.

Figura 13 – Protótipo da coleira de monitoramento



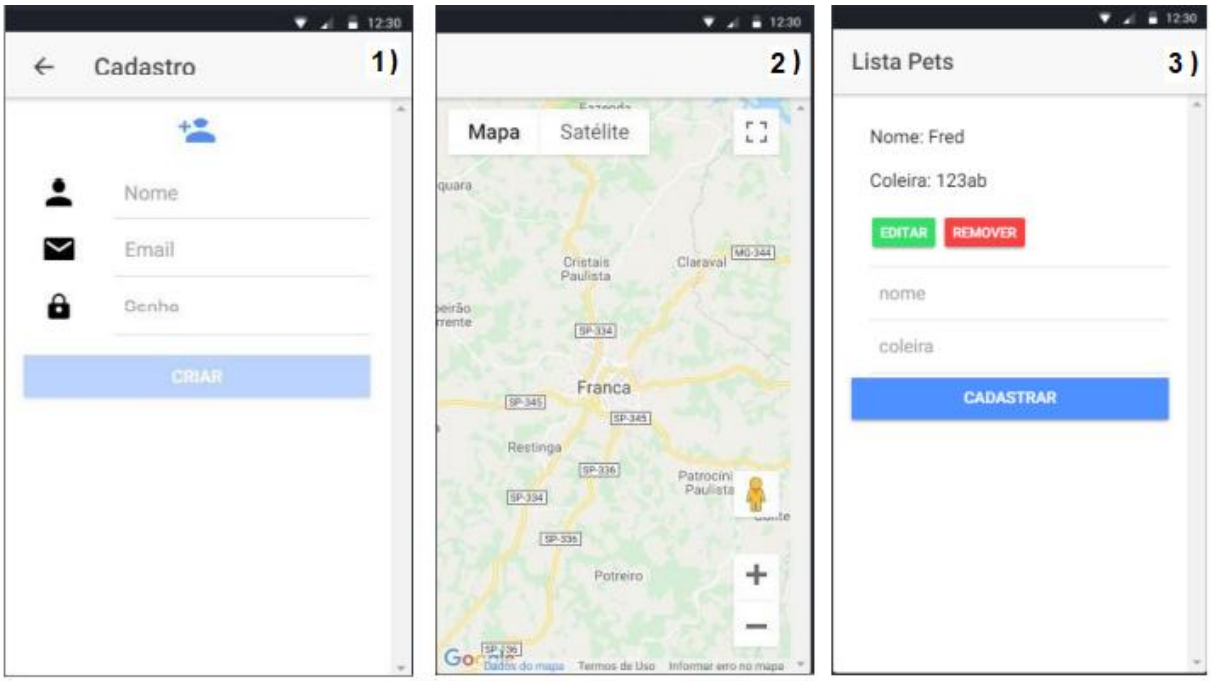
Fonte: Adaptado pelo autor com base em SILVA, FEYH e ROLAND (2018).

Para o desenvolvimento da aplicação *mobile*, é utilizado banco de dados Firebase e o *framework* Ionic 3 que permite a criação de aplicativos híbridos (GOIS, 2017). Segundo Silva, Feyh e Roland (2018), para desenvolver o software foi criado

um diagrama *Unified Modeling Language* (UML), um diagrama *Business Process Model and Notation* (BPMN), um estudo de casos de uso e a prototipação das telas.

Ao entrar no aplicativo, o primeiro passo é realizar o *login* ou cadastro, caso ainda não tenha. O item 1 da Figura 14 mostra a tela de *login*. Em seguida, o aplicativo faz uma verificação no banco de dados para ver se o cadastro já existe e, caso não exista, trata de realizar a criação. Na sequência, o aplicativo direciona para a tela principal denominada *Home*, que exibe o Google Maps e a localização das coleiras cadastradas como mostra o item 2. Na tela de cadastro de coleiras o usuário pode incluir seus animais e o respectivo número da coleira, que serve como um indicador conforme indica o item 3.

Figura 14 – Tela de *login*, mapa e cadastro de pets



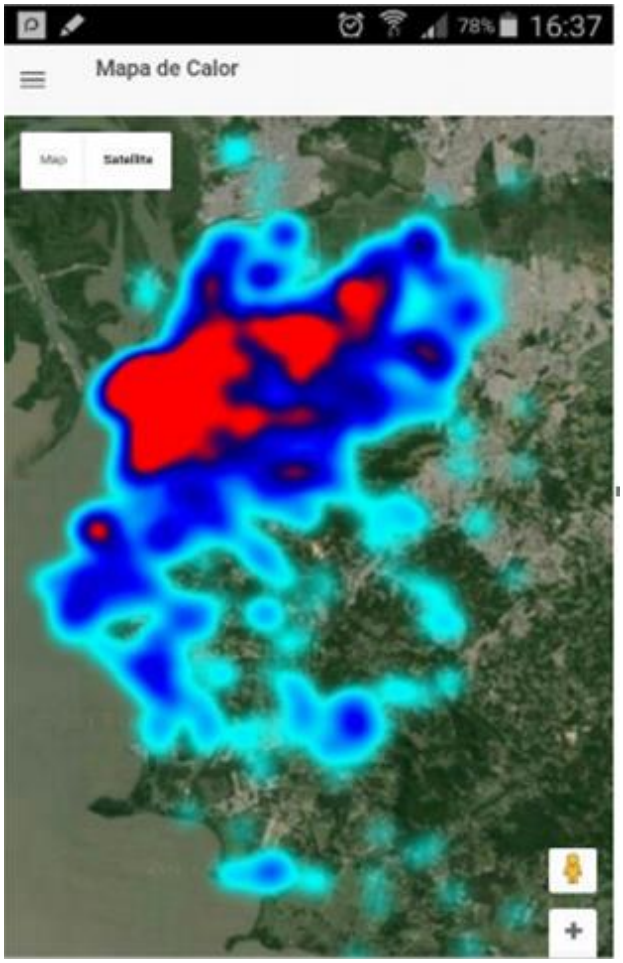
Fonte: Adaptado pelo autor com base em SILVA, FEYH, ROLAND (2018).

3.4 Sistema gerador de rotas através de mapas de calor

O presente trabalho foi desenvolvido por um aluno da Universidade do Vale do Taquari – UNIVATES, em 2015, com o objetivo de mapear áreas com maiores índices de criminalidade e traçar rotas para as viaturas da polícia trafegarem por estes locais, desta forma reprimindo o acontecimento de novos incidentes.

Para mapeamento das regiões com maiores índices de criminalidade, os Boletins de ocorrência (BO) registrados no Banco de dados da Polícia Civil são considerados. Eles são consultados através de *web service* do tipo *Representational State Transfer* (REST) e para a geração do mapa de calor que demonstra a criticidade da criminalidade das regiões, é utilizado o Google Maps API Java Script V3. O resultado pode ser visto na Figura 15.

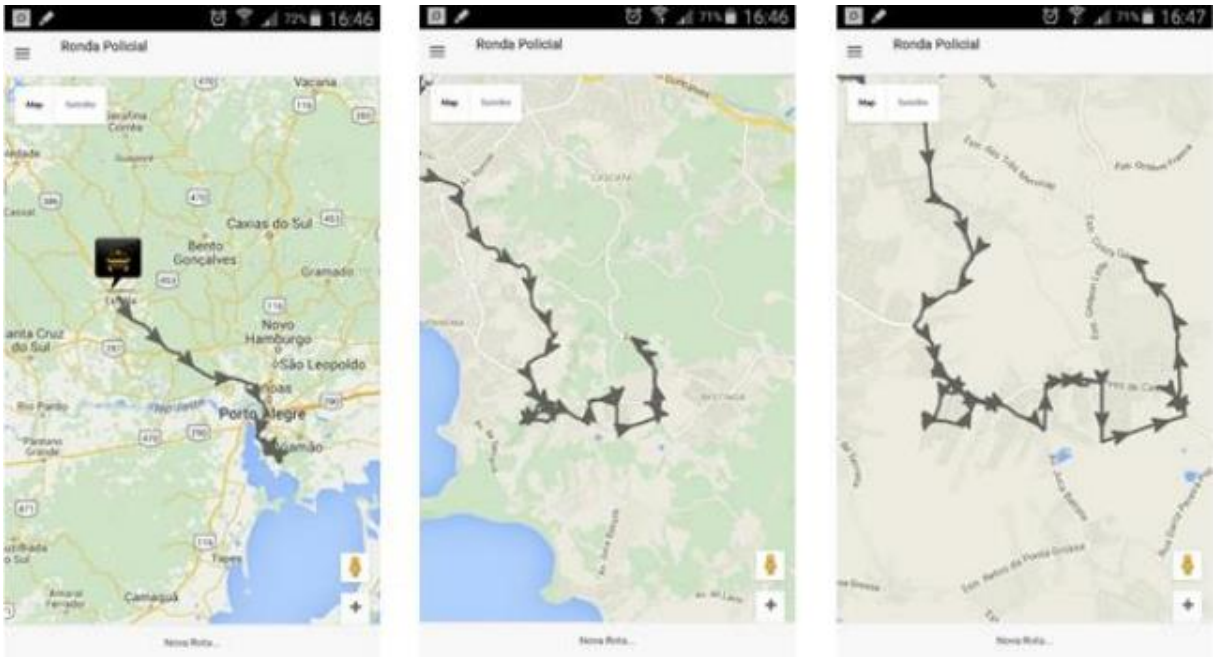
Figura 15 – Exemplo de visualização do mapa de calor



Fonte: Adaptado pelo autor com base em DANIELI (2015).

Em relação a geração das rotas, que está ilustrada na Figura 16, é utilizada a Google JavaScript Directions API, que possibilita criar rotas dentro da malha viária do mapa, podendo definir a origem, destino e pontos de passagem.

Figura 16 – Exemplo de rotas geradas para o usuário logado



Fonte: Adaptado pelo autor com base em DANIELI (2015).

Quanto ao armazenamento e persistência de dados, foi utilizado o banco de dados PostgreSQL juntamente do módulo PostGIS que possibilita adicionar entidades geográficas ao PostgreSQL. Através dele é possível usar objetos *Geographic Information System* (GIS) e consequentemente funções espaciais e topológicas. Para expressar dados espaciais são usados objetos como POINT (um ponto, composto por duas coordenadas), LINESTRING (uma linha composta por quatro pontos, onde dois deles representam o ponto de partida e os outros dois o ponto de chegada), POLYGON (um polígono), MULTIPOINT (vários pontos), MULTILINESTRING (várias linhas) e MULTIPOLYGON (vários polígonos).

3.5 Desenvolvimento de sistema de geolocalização e rastreamento para a plataforma Android – COMPASS

Desenvolvido por um aluno da Universidade Tecnológica Federal do Paraná, o presente trabalho tratou de desenvolver uma solução *mobile* para Android capaz de mapear a localização geográfica dos clientes de uma determinada empresa e

fornecer formas mais eficazes para programar as visitas feitas pelos representantes comerciais.

Além disso, a solução também permite que a empresa possa medir a distância exata percorrida pelos seus representantes, visto que normalmente as empresas baseiam-se apenas em tabelas de distância entre um ponto e outro, sem levar em consideração a rota percorrida. A solução também permite ter certeza de que todos os clientes foram visitados, uma vez que as rotas ficam armazenadas em um banco de dados.

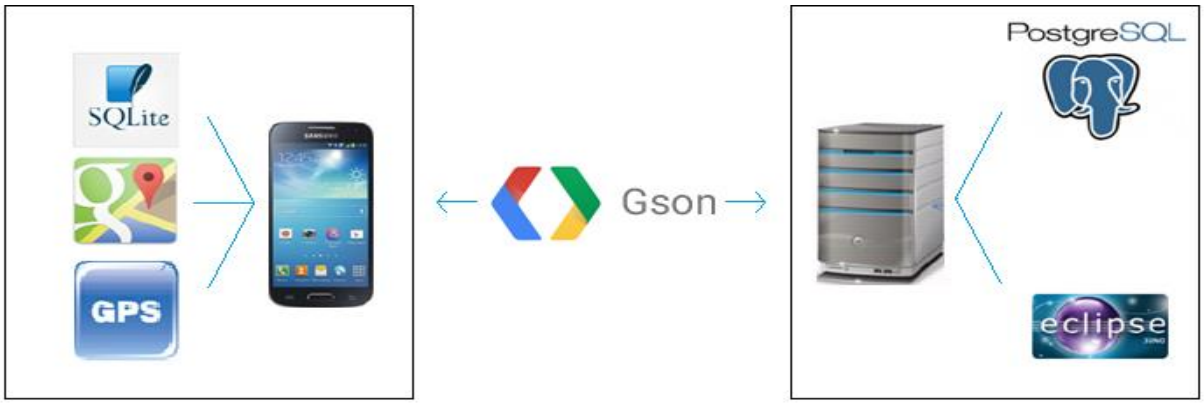
O sistema desenvolvido neste trabalho é integrado ao banco de dados de um *Enterprise Resource Planning* (ERP) por meio da rede, através de *sockets.* Desta forma, é possível ter acesso aos dados dos clientes da empresa e ao mesmo tempo sincronizar os dados referentes as rotas.

Ao entrar no aplicativo, ele inicia a leitura e gravação dos pontos geográficos percorridos. A leitura da posição é programada para ser feita a cada poucos segundos e somente se houve mudança maiores que 20 metros desde a última leitura. Ao gravar a leitura, a data e hora também são gravadas para que seja possível reconstruir o caminho percorrido.

O desenvolvimento foi feito através do Eclipse Juno IDE e Android SDK, possibilitando desenvolver o aplicativo para Android versão 4.1. Para realizar a persistência de dados no aparelho Android foi utilizado o SQLite, enquanto o banco de dados no servidor, que atua como repositório central de dados, é o PostgreSQL 9.3.

A comunicação entre cliente e servidor é feita através de *socket* e para transporte das informações são utilizados objetos da biblioteca GSON versão 2.3.1 conforme ilustra a Figura 17:

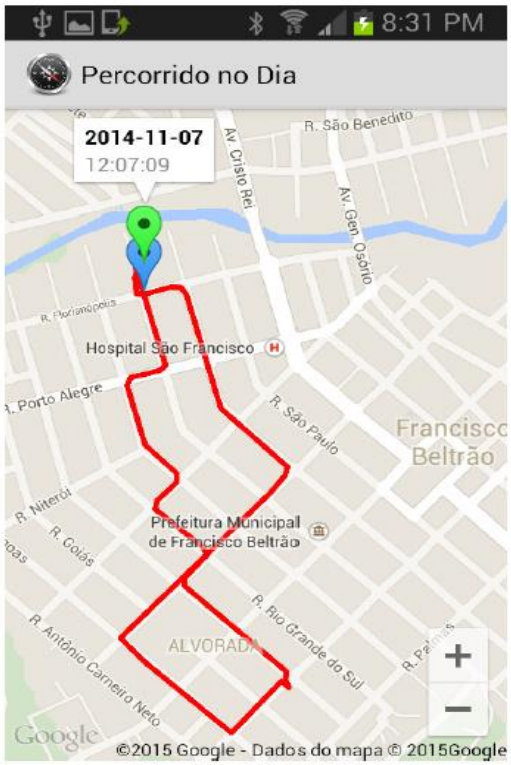
Figura 17 – Arquitetura da solução proposta



Fonte: Adaptado pelo autor com base em MACHADO (2015).

Em relação a visualização do mapa, é utilizado o Google Maps API. Para marcação da localização dos clientes no mapa são utilizados marcadores *Markers*. Para visualização do caminho percorrido pelo representante, a classe *PolygonOptions* é utilizada, traçando uma linha contínua que liga todos os pontos gravados pelo rastreamento do GPS. A Figura 18 mostra um mapa com dois *Markers* e o caminho percorrido em vermelho.

Figura 18 – Mapa utilizando Markers e Rota



Fonte: Adaptado pelo autor com base em MACHADO (2015).

3.6 Comparativo entre os trabalhos relacionados

Através da Tabela 4 é possível fazer um comparativo entre os principais aspectos dos trabalhos relacionados e a proposta do presente trabalho.

Tabela 4 – Comparativo entre os trabalhos relacionados

Característica	**TRAB. 3.1**	**TRAB. 3.2**	**TRAB. 3.3**	**TRAB. 3.4**	**TRAB. 3.5**	**Proposta**
Plataforma	Android, iOS e Web	Android e iOS	Híbrido	Android	Android	Android
Possui chat	Sim	Sim	Não	Não	Não	Sim
Tipo de Armazenamento	Não espefic.	Não espefic.	Firebase	PostgreSQL + PostGIS	SQLite	Firebase
Pago	Sim	Sim	Não	Não	Não	Não
Permite ocultar localização	Sim	Não	Não	Não	Não	Sim
Possui dinâmica de grupos	Sim	Sim	Não	Não	Não	Sim

Fonte: Do autor (2020).

Constata-se que, embora a variedade de aplicações para rastreamento seja grande no mercado *mobile*, o aplicativo proposto possui alguns diferenciais dos demais, tanto na usabilidade quanto na parte técnica. Sendo assim, o capítulo 4 descreve os métodos e as tecnologias que serão utilizadas para elaboração do aplicativo proposto.

4 MÉTODOS E TECNOLOGIAS

Este capítulo apresenta as tecnologias utilizadas no desenvolvimento do aplicativo, bem como os métodos utilizados na elaboração do trabalho.

Para Lakatos e Marconi (2001), no método dedutivo parte-se do entendimento geral e aplica-se a um caso particular. Assim sendo, o presente trabalho analisa as melhores práticas e soluções de rastreamento geográfico em âmbito global, adaptando essas soluções com as necessidades no aplicativo proposto.

A pesquisa realizada é abordada de maneira quantitativa e qualitativa. De acordo com Prodanov e Freitas (2013), a pesquisa quantitativa faz uso de estatísticas e números para mensurar aspectos e correlacionar situações. Já o modo qualitativo é caracterizado por ser mais simples e aberto a interpretação e análise do autor mediante os dados. A pesquisa se encaixa no âmbito quantitativo pois é realizado um estudo sobre a popularidade de *smartphones*, acesso à *internet* e serviços de telefonia, visando comprovar a viabilidade da solução proposta. No âmbito qualitativo, a pesquisa se encaixa dado que o aplicativo é remetido para validação e sua qualidade é medida através de determinadas métricas.

O presente trabalho emprega pesquisa de caráter exploratório e descritivo. Para Gil (2002, p. 41), a pesquisa exploratória visa proporcionar familiaridade com o problema seu objetivo principal é o aprimorar ideias ou descobrir intuições. Já a pesquisa descritiva busca descrever fenômenos, características ou experiências sobre o estudo realizado (PRODANOV; FREITAS, 2013). No presente trabalho, a pesquisa exploratória está relacionada com os objetivos de compreender as técnicas

de localização existentes nos dispositivos móveis, conhecer aplicativos de rastreamento geográfico disponíveis no mercado e avaliar trabalhos relacionados ao tema. A pesquisa descritiva está relacionada com o conjunto de informações obtidas e documentadas durante o desenvolvimento do trabalho.

Os procedimentos adotados no presente trabalho são pesquisa bibliográfica, experimental e pesquisa-ação. Segundo Prodanov e Freitas (2013), a pesquisa bibliográfica visa aproximar o pesquisador ao material relacionado ao assunto da pesquisa, enquanto na pesquisa-ação o pesquisador tem um envolvimento maior, dado que o objetivo é solucionar, acompanhar e avaliar o problema relacionado. A pesquisa experimental consiste em observar o comportamento de um objeto de estudo influenciado por variáveis manipuladas e tem como finalidade testar hipóteses e aprender relações de causa e efeito (SILVEIRA, 2019).

A pesquisa bibliográfica foi utilizada no embasamento teórico necessário para compreender os principais aspectos relacionados ao trabalho, como os sistemas de localização, arquitetura da plataforma Android e serviços BAAS. A pesquisa-ação foi utilizada durante a elaboração do trabalho, onde mapeou-se os problemas e situações específicas que exigiram determinados tratamentos para alcançar os objetivos propostos. Por fim, a pesquisa experimental está ligada ao objetivo de conhecer aplicativos de rastreamento geográfico existentes no mercado, trabalhos relacionados ao tema e o funcionamento dos sistemas de localização na prática.

4.1 Tecnologias

As seções abaixo descrevem as tecnologias utilizadas no desenvolvimento do aplicativo proposto.

4.1.1 Firebase Authentication

O Firebase Authentication é uma ferramenta que permite realizar autenticação, criação de contas de usuários e gerenciamento de seções de *login*,

além de oferecer suporte para redefinição de senha por *e-mail*. As autenticações podem ser feitas através de *e-mail* e senha ou contas federadas ao Google, como o Facebook, Gmail, Twitter, entre outras. A criação de contas permite acessar dados básicos como nome, *e-mail* e foto de perfil, além de que a base de dados pode ser facilmente compartilhada com outros projetos caso o desenvolvedor ache necessário (DOCUMENTATION FIREBASE AUTH, 2020).

Seu funcionamento consiste em enviar as credenciais ao SDK do Firebase e aguardar o serviço *backend* processar a informação e enviar uma resposta ao dispositivo. Todo este fluxo é feito através de tarefas assíncronas para dar sensação de responsividade ao usuário (CAMILO, 2020).

4.1.2 Firebase Cloud Firestore

O Cloud Firestore é uma ferramenta que permite criar bancos de dados não relacionais orientados a documento, onde a hospedagem dos dados acontece na nuvem. Os documentos são escritos na sintaxe JSON e compostos por pares de chave-valor. Eles são armazenados em coleções e cada coleção também pode ter diversas outras coleções atreladas, formando desta forma uma estrutura de árvore (PONTES, 2019).

De acordo com o Google, o Cloud Firestore é uma ferramenta baseada no Realtime Database, porém com inúmeras melhorias, como consultas mais eficientes, melhor escalabilidade e modelo de dados mais intuitivo (DOCUMENTATION FIREBASE FIRESTORE, 2020).

Um aspecto relevante para o desenvolvimento deste trabalho é que o Cloud Firestore também oferece suporte para persistência de dados *offline.* Desta forma, mesmo que o aparelho não tenha conexão com a *internet*, é possível trabalhar com os dados que já foram sincronizados. Quando a conexão é reestabelecida, todas as alterações são sincronizadas com a nuvem. A Figura 19 traz uma ilustração sobre o funcionamento do Cloud Firestore, onde um usuário insere registros no banco de dados e eles são imediatamente sincronizados com outros dispositivos.

Figura 19 – Cloud Firestore

Fonte: Adaptado pelo autor com base em DOCUMENTATION GOOGLE FIRESTORE (2020).

Um dos recursos do Cloud Firestore empregado neste trabalho é o *Snapshot Listene*r, cuja função é sincronizar atualizações de documentos em tempo real. Conforme a documentação do Google, o recurso permite criar um *snapshot* dos documentos e monitorá-los através de um *listener* sincronizado com a nuvem. Caso o conteúdo de algum destes documentos for alterado, tanto na nuvem quanto localmente, uma chamada trata de sincronizá-lo imediatamente (DOCUMENTATION ANDROID SNAPSHOT LISTENER, 2020).

Este recurso torna-se ideal para realizar a atualização das posições de cada membro no mapa e na implementação do *chat*, uma vez que o Cloud Firestore faz um *callback* sempre que um dado for atualizado, não sendo necessário implementar rotinas de verificação periódicas para proporcionar ao usuário a sensação de tempo real.

No presente trabalho, os eventos de inserção dos *Snapshot Listeners* também foram implementados para acionarem uma *trigger* do Firebase Cloud Functions, que por sua vez é encarregada de acionar o Firebase Cloud Messaging para exibição de notificações *push* ao usuário quando mensagens forem escritas no *chat.*

4.1.3 Firebase Cloud Messaging

O Firebase Cloud Messaging possibilita o envio de mensagens e notificações gratuitamente entre plataformas de maneira rápida e confiável. É possível enviá-las para usuários específicos, para um grupo de dispositivos ou para usuários inscritos em algum determinado tópico. No presente trabalho, este recurso é utilizado para gerar notificações *push* quando novas mensagens forem enviadas nos *chats*.

A implementação é dividida em duas partes, sendo elas *client-side* e *server-side*. Do lado do cliente, a lógica para recebimento de mensagens e exibição de notificações é implementada, enquanto no lado do servidor implementa-se o *backend* para geração dos *payloads* (DOCUMENTATION FIREBASE CLOUD MESSAGING, 2020).

4.1.4 Firebase Cloud Functions

Firebase Cloud Functions é de um *framework* sem servidor que possibilita a execução de códigos de *backend* em resposta a eventos acionados por outros recursos do Firebase, como o Cloud Firestore. No presente trabalho, o código *backend* foi escrito em Node.js e seu acionamento ocorre quando uma mensagem for incluída na coleção de mensagens dos grupos (DOCUMENTATION FIREBASE CLOUD FUNCTIONS, 2020).

4.1.5 Google Maps API

O Google Maps é um mapa digital desenvolvido pelo Google que cobre 99% do território mundial, abrange 200 países, conta com cerca de 25 milhões de atualizações por dia e possui 1 bilhão de usuários ativos (DOCUMENTATION GOOGLE MAPS API, 2020).

Um conjunto de APIs é disponibilizado pelo Google para trabalhar com estes mapas, possibilitando calcular rotas, pesquisar endereços, incluir marcadores,

interagir com o mapa, entre outras funcionalidades (SOUSA, 2016). Essas APIs são divididas em três categorias: *Maps, Routes* e *Places* (DOCUMENTATION GOOGLE MAPS API, 2020).

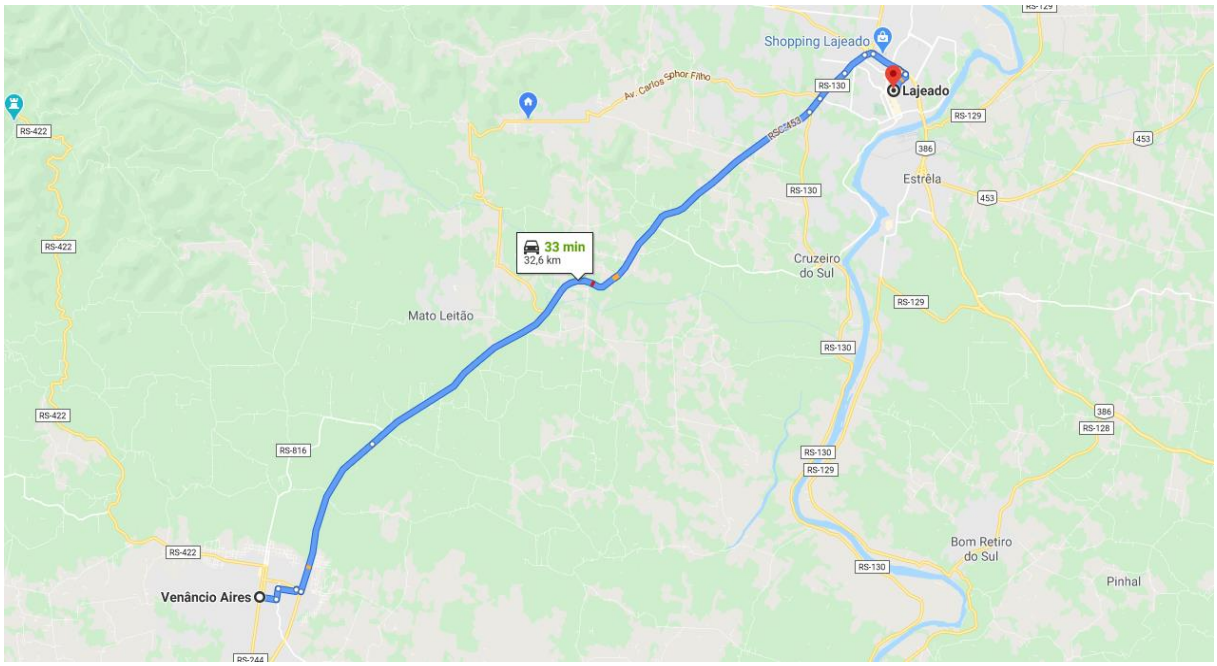
Maps API: permite adicionar mapas escalonados em aplicações Web e *mobile*, sendo eles interativos ou estáticos. A visualização do mapa pode ser feita em 360º, com opção de *zoom in* e *zoom out*, além de ser possível configurar o estilo do mapa, adicionar linhas, cores, imagens e marcações.

Figura 20 – Exemplo mapa

Fonte: Adaptado pelo autor com base em DOCUMENTATION AGOOGLE MAPS API (2020).

Routes API: permite a criação e exibição de rotas confiáveis, auxiliando os usuários a realizarem seus percursos com transporte público, bicicleta, carro ou a pé. Essas rotas são frequentemente atualizadas e cobrem 64 milhões de quilômetros de estradas em mais de 200 países e territórios. Através desta API também é possível medir a distância entre pontos, o tempo aproximado de deslocamento e a criação de itinerários.

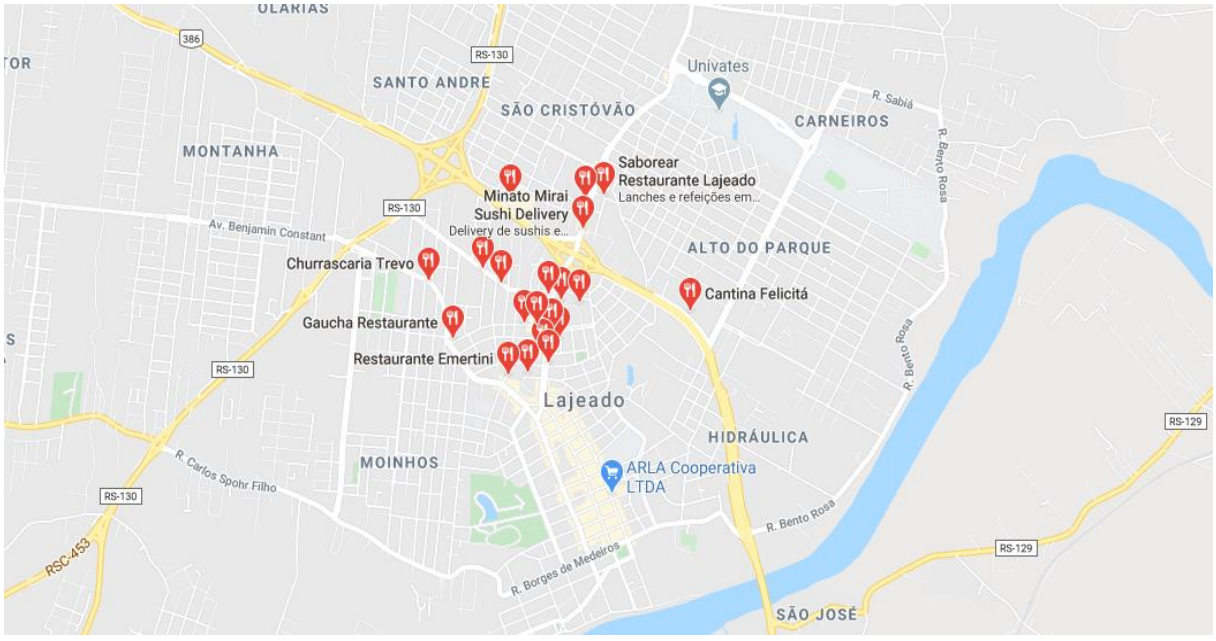
Figura 21 – Exemplo rotas



Fonte: Adaptado pelo autor com base em DOCUMENTATION AGOOGLE MAPS API (2020).

Places API: permite obter a localização precisa de estabelecimentos comerciais, além de retornar informações sobre o ambiente, avaliações de outros usuários, classificações e dados de contato. A API atualmente conta com dados sobre mais de 150 milhões de lugares e pontos de interesse.

Figura 22 – Exemplo lugares



Fonte: Adaptado pelo autor com base em DOCUMENTATION AGOOGLE MAPS API (2020).

De acordo com as políticas de venda do Google, até um determinado número de visualizações a utilização das APIs não há nenhum custo. A partir deste número de visualizações o serviço se torna pago. Entretanto, o próprio Google afirma que para a maioria dos usuários o número máximo de visualizações é suficiente (DOCUMENTATION GOOGLE MAPS API, 2020).

4.1.6 Android Studio

Android Studio é a principal plataforma de desenvolvimento de *softwares* para a plataforma Android. Trata-se de uma IDE oficial do Google, com linguagem JAVA e com um ambiente de desenvolvimento integrado, que permite realizar a depuração do aplicativo em tempo real enquanto ele é executado. Além disso, para facilitar a realização dos testes por parte do desenvolvedor, o Android Studio também disponibiliza um simulador nativo que permite emular diversas versões do sistema Android (DE ARAÚJO, 2018).

4.1.7 AlarmManager

AlarmManager é uma classe nativa do Android que permite agendar rotinas de execução em segundo plano em intervalos de tempos regulares ou irregulares (DOCUMENTATION ANDROID ALARMMANAGER, 2020). Ela torna-se necessária para o desenvolvimento do presente trabalho pois, mesmo que o aplicativo não esteja em execução, é preciso executar rotinas para capturar as coordenadas geográficas e sincronizá-las com o Cloud Firestore.

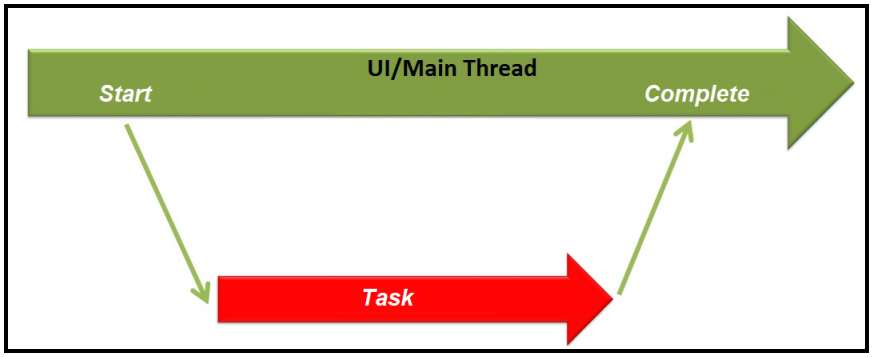
4.1.8 Task assíncrona

A necessidade de executar tarefas assíncronas durante a execução do aplicativo é importante para não sobrecarregar a *main thread* e desta forma dar ao usuário a sensação de responsividade. Por exemplo, tarefas que dependem de uma

resposta de algum servidor, como *login* ou comunicação com banco de dados podem causar sensação de travamento ou lentidão.

Uma abordagem para resolver este problema é executar estas tarefas de forma assíncrona. Para isso a plataforma Android disponibiliza a API Task, cujo objetivo é possibilitar que a *thread* principal dispare uma tarefa e receba um *callback* quando chegar ao fim, processo este que pode ser visto na Figura 23 (ANDROID DOCUMENTATION TASK, 2020).

Figura 23 – Task assíncrona



Fonte: Adaptado pelo autor com base em ANDROID DOCUMENTATION TASK (2020).

5 DESENVOLVIMENTO

O presente trabalho tratou sobre o desenvolvimento de um aplicativo para dispositivos móveis que utilizam a plataforma Android, sendo um dos objetivos possibilitar que os usuários compartilhem suas localizações geográficas em tempo real com outras pessoas. Para conectá-los foi utilizada a dinâmica de grupos, onde os membros participantes possuem acesso em tempo real sobre a localização dos outros membros e mensagens do *chat*. Qualquer usuário pode criar grupos, sendo eles públicos ou protegidos por senha. Também é possível compartilhar a chave de acesso do grupo via Whatsapp, e-*mail*, Facebook e afins.

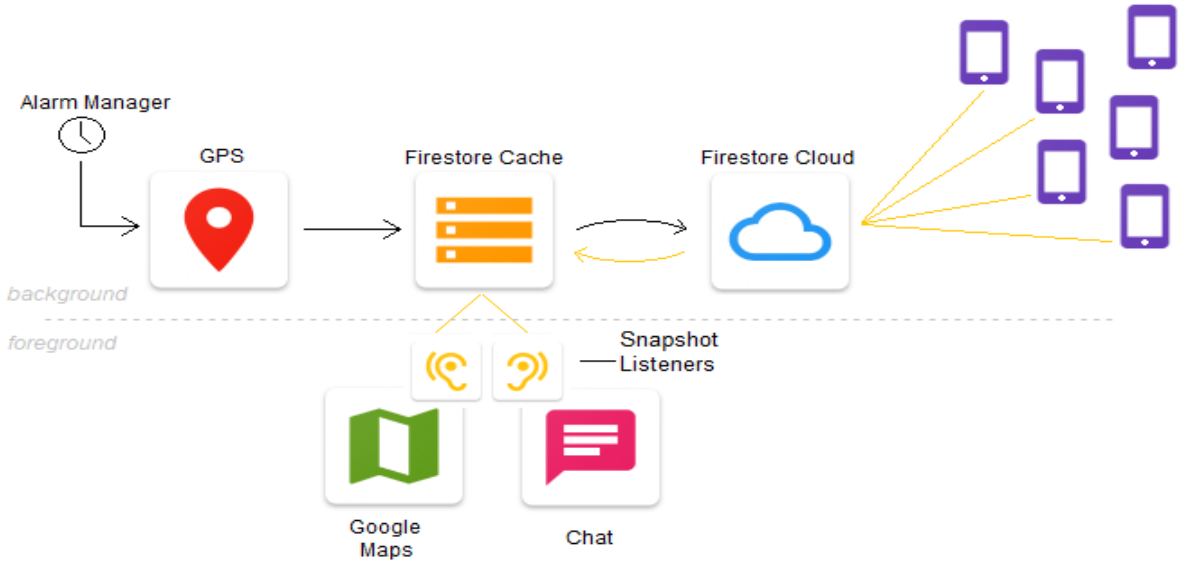
Tendo em vista que nem sempre o dispositivo terá conexão com a *internet*, será necessário fazer o uso de recursos para armazenar os dados localmente e sincronizá-los com a nuvem quando a conexão for reestabelecida. E levando em conta que o aplicativo nem sempre será executado em primeiro plano, torna-se necessário implementar rotinas para serem executadas em segundo plano, visando capturar a localização geográfica e sincronizá-la com a nuvem periodicamente.

As seções a seguir detalham como a implementação foi feita para atender aos requisitos, trazendo detalhes sobre a arquitetura, diagrama de estados, estruturação do banco de dados, implementação e testes.

5.1 Arquitetura

A arquitetura do aplicativo foi projetada para capturar as coordenadas geográficas dos serviços de localização do dispositivo e imediatamente sincronizá-las com a nuvem. Essa tarefa é executada periodicamente em segundo plano pelo AlarmManager, uma vez que o aplicativo não estará em execução o tempo inteiro. Outro cuidado em sua elaboração é que eventualmente o dispositivo pode não ter conexão com a *internet* e neste caso as coordenadas obtidas pelo sistema de localização precisam ser armazenadas *offline* e sincronizadas com a nuvem quando a conexão for reestabelecida. Para isso será utilizado o Cloud Firestore que oferece suporte para armazenamento de dados *offline* e na nuvem. A sincronização destes dados é feita através de métodos do próprio Cloud Firestore e requer conexão com a *internet*. Para compreender a arquitetura do aplicativo Finded é possível observar a Figura 24:

Figura 24 – Arquitetura do aplicativo



Fonte: Do autor (2020).

Durante a execução em segundo plano o aplicativo obtém a localização do dispositivo e armazena esse dado localmente (*cache*), possibilitando que seja sincronizado com a nuvem quando houver conexão com a *internet*. Todo este processo é repetido periodicamente pelo AlarmManager. Quando o aplicativo estiver sendo executado em primeiro plano, os dados como coordenadas dos membros e

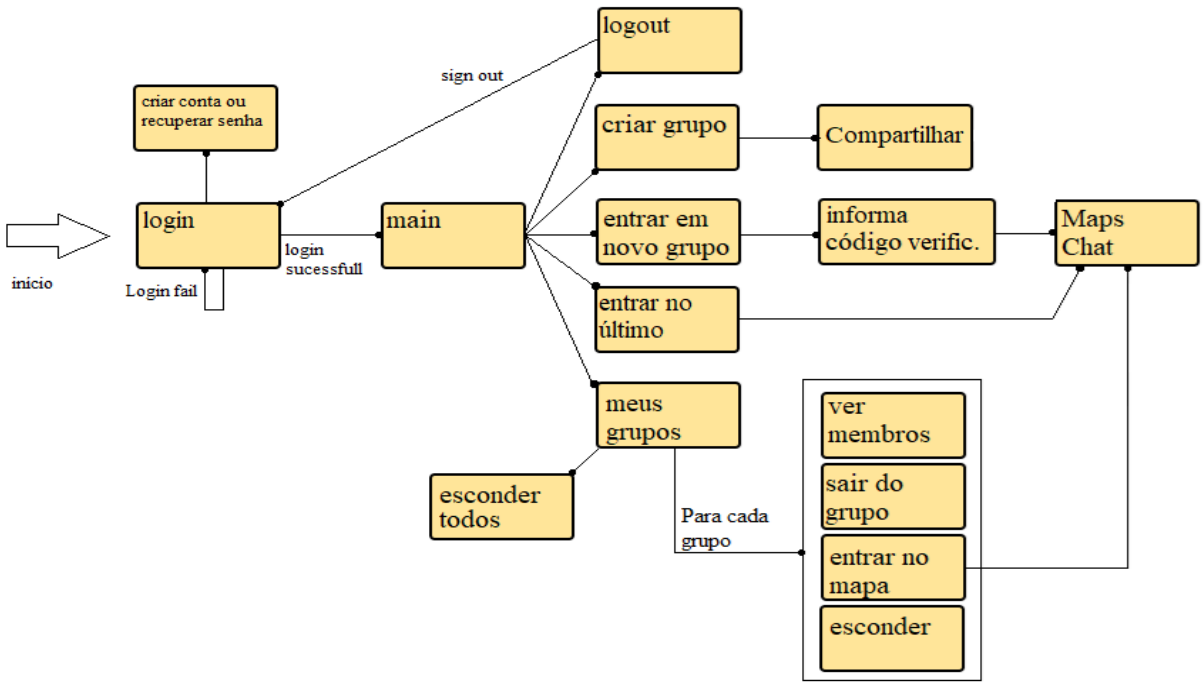
mensagens do *chat* serão exibidos para o usuário, enquanto os *Snapshot Listeners* serão ativados para receber atualizações destes dados em tempo real. Embora o mapa só exiba a última posição de cada usuário, a rota percorrida ficará armazenada internamente no Cloud Firestore, dentro de uma coleção específica.

Para um melhor entendimento, a seção 5.4.3 descreve detalhadamente como as rotinas de execução em segundo plano se comportam e a seção 5.4.4 demonstra o comportamento adotado em relação a persistência de dados em *cache* e na nuvem.

5.2 Diagrama de estados

Para elucidar as principais ações disponíveis ao usuário é feito o uso de máquina de estados, como mostra a Figura 25. Para compreender tais eventos é essencial ter conhecimento sobre os casos de uso da ferramenta, que neste caso consiste em criar, entrar ou sair de grupos e ter acesso sobre as informações de cada grupo, como os membros e as mensagens do *chat.*

Figura 25 – Ações disponíveis ao usuário



Fonte: Do autor (2020).

O primeiro passo ao abrir o aplicativo é realizar o *login* ou criar uma conta. Em seguida o usuário será direcionado para a tela principal do aplicativo onde terão cinco opções disponíveis. A primeira delas permite realizar o *logout,* redirecionando o usuário para a tela de *login* novamente. A segunda opção permite criar um grupo e convidar outros usuários por mensagem via Whatsapp e afins.

A terceira opção permite ingressar em um novo grupo onde é necessário informar o seu código de verificação e senha, se for privado. Feito isso, o usuário será direcionado para a Activity com o mapa e o *chat* daquele grupo. A quarta opção é apenas um atalho para acessar o último grupo sem precisar listá-lo antes.

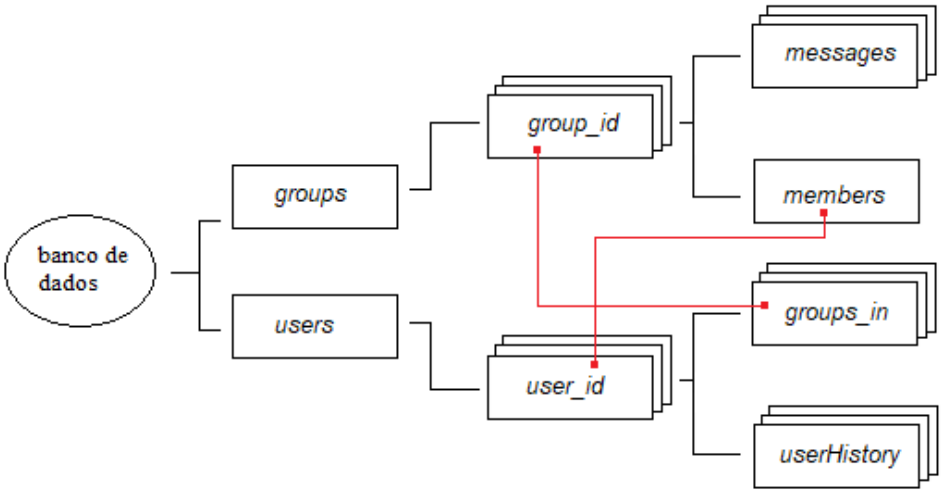
A quinta opção leva o usuário para uma tela onde são listados todos os grupos que ele faz parte. Para cada um destes grupos listados, o usuário pode fazer quatro ações: a primeira ação irá listar informações sobre os membros do grupo e a segunda opção permitirá sair do grupo. A terceira permite acessar o mapa e *chat* do grupo e a quarta opção permite ativar o recurso de invisibilidade, interrompendo o envio de localizações geográficas para os membros do grupo selecionado.

Por último, uma opção chamada “esconder todos” permitirá que o usuário fique invisível para todos os grupos de uma só vez, evitando que ele tenha que entrar em cada um dos grupos para fazer isso.

5.3 Banco de dados

Tendo em vista a utilização do Firebase Cloud Firestore para armazenamento de dados, que consiste em um banco de dados não relacional e orientado a documentos, a estruturação foi feita conforme mostra a Figura 26, onde cada documento é escrito na sintaxe JSON e composto por chaves de *pares-valor*.

Figura 26 – Estrutura do banco de dados



Fonte: Do autor (2020).

Nota-se que há duas coleções principais para segmentar informações dos grupos e dos usuários. Dentro da coleção de usuários há diversos documentos identificados pelo campo *user_id* e, dentro destes documentos existem duas coleções para armazenar a lista de grupos que ele faz parte (*groups_in)* e seu histórico de localização (*userHistory).* A coleção de grupos possui uma lógica semelhante, onde cada grupo possui uma identificação (*group_id)* e dentro destes documentos existem duas coleções para armazenar a lista de membros (*members)* e as mensagens do *chat* (*messages).*

Apesar do banco de dados não ser relacional, é estabelecida uma referência entre os documentos, como é o caso do *user_id* com *members* e do *group_id* com *groups_in.* Este tipo de relacionamento é utilizado na execução de consultas compostas, as quais estão descritas na seção 5.4.4.

5.4 Implementação

Após a definição da arquitetura do sistema, diagrama de estados e estruturação do banco de dados, deu-se início ao desenvolvimento do aplicativo. Cada etapa é abordada nas subseções seguintes.

5.4.1 Autenticação

Através do Firebase Authentication foram implementadas opções para realizar autenticação de usuários, criação de novas contas, recuperação de senha por *e-mail* e gerenciamento de seções de *login* dentro do aplicativo. Para isso, no console do Firebase, os métodos de *login* por *e-mail*/senha e conta Google foram habilitadas e em seguida as dependências do *FirebaseAuth 19.1.0* e *Services Auth 17.0.0* foram importadas no aplicativo. A primeira delas permite realizar autenticação de usuários, criação de novas contas e gerenciamento de sessões de *login*. A segunda, o *login* através da conta Google.

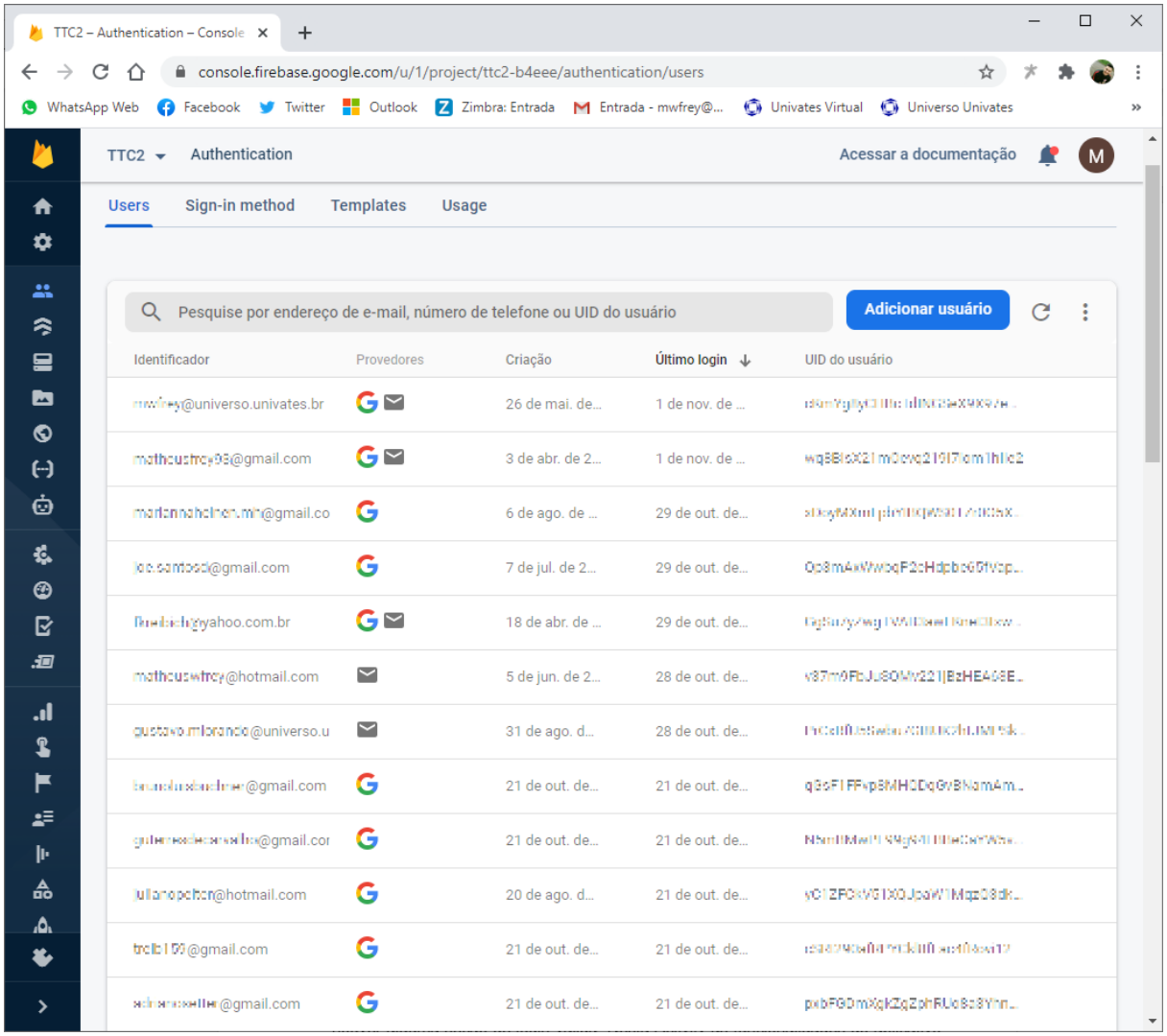
Para realização de *login* através de *e-mail* e senha, primeiramente é necessário cadastrar o usuário na base de dados do Firebase Authentication, tarefa feita pelo método *createUserWithEmailAndPassword* cujos únicos parâmetros são o *e-mail* e senha. Por questões de segurança também é feito a utilização de um método complementar que envia um e-mail de verificação para o usuário que estiver se cadastrando, denominado *sendEmailVerification.* Uma vez que a conta foi criada, basta usar o método *signInWithEmailAndPassword* e aguardar o *callback* para verificar se a autenticação foi realizada com sucesso. Caso a autenticação tenha falhado uma mensagem é exibida para o usuário, como por exemplo, senha inválida, conta inexistente, conta não verificada, entre outras possíveis causas.

Para realização de *login* através de conta federada ao Google, utilizou-se a *intent* do Google Services para abrir a tela que permite selecionar a conta e o método *onActivityResult* para tratar o resultado da autenticação. Diferente da autenticação por *e-mail* e senha, no qual o próprio aplicativo manipula as credenciais, neste formato a *intent* apenas retorna a credencial *OAuth* que é enviada para o método *signInWithCredential*, encarregado de aprovar ou rejeitar a autenticação requerida.

O mesmo usuário pode autenticar-se de ambas as maneiras, desde que seja com o mesmo *e-mail*. Nestes casos o Firebase Authentication irá atribuir uma única identificação para este usuário. Essa identificação é essencial para identificar cada usuário dentro do aplicativo desenvolvido. A Figura 27 apresenta um exemplo de

usuários que já se autenticaram no aplicativo até o presente momento, onde é possível ver que alguns deles usaram provedores diferentes:

Figura 27 – Exemplo usuários autenticados



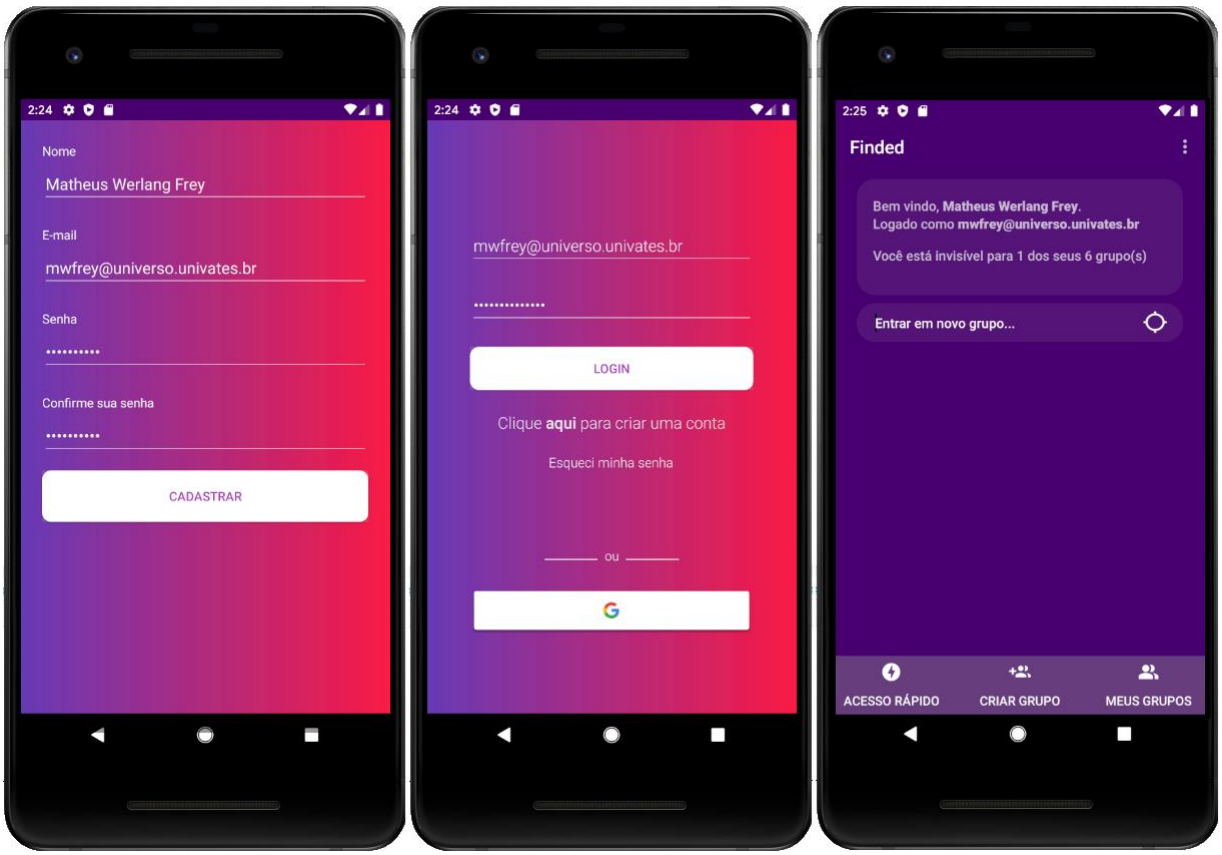
Fonte: Do autor (2020).

Para controlar o fluxo de autenticação foram criadas três Activities como indica a Figura 28. A primeira delas permite criar contas, onde é necessário informar o nome, *e-mail* e senha. Para concluir o cadastro é enviado um *link* de confirmação para o *e-mail* informado.

A segunda Activity permite efetuar *login* através de *e-mail*/senha ou através da conta Google. Nela também é disponibilizado um recurso para redefinição de senha, onde um *link* é enviado para o *e-mail* do usuário.

A terceira é a Activity principal que será aberta somente se houver alguma seção de *login* válida. Nesta Activity todas as funcionalidades do aplicativo estão acessíveis ao usuário.

Figura 28 – Fluxo de autenticação



Fonte: Do autor (2020).

Por padrão ao abrir o aplicativo, a Activity principal sempre é iniciada e no método *onStart* uma verificação é realizada para ver se há seção de *login* ativa. Caso não tenha, o usuário simplesmente é direcionado para a tela de *login* e a Activity principal é destruída.

Caso tenha seção ativa, uma série de rotinas são executadas para carregar as informações do usuário logado, apresentar informações sobre os grupos que ele faz parte, requisitar acesso aos recursos do dispositivo, executar alarmes para execução em segundo plano, entre outras tarefas necessárias para o correto funcionamento do aplicativo.

Figura 29 – Verificação de seção de *login*

```
171     @Override
172     public void onStart() {
173         super.onStart();
174         // Check if user is signed in (non-null) and his mail is verified
175         currentUser = mAuth.getCurrentUser();
176         if (currentUser != null && currentUser.isEmailVerified()) {
177             searchUser(mAuth.getUid());
178             ignoreBatteryOptimatizion();
179         } else {
180             Intent intent = new Intent( packageContext: MainActivity.this, LoginActivity.class);
181             startActivity(intent);
182             finish();
183         }
```

Fonte: Do autor (2020).

Na Figura 29 é possível observar a verificação desenvolvida, onde é feito um teste para ver se o usuário retornado pela instância do FirebaseAuth é diferente de nulo e se o seu *e-mail* está verificado. Caso os dois testes forem satisfatórios, então o método *searchUser* trata de iniciar as rotinas necessárias para inicialização do aplicativo. Do contrário, o usuário é redirecionado para tela de *login* e a Activity principal é destruída.

Também foram implementadas opções para realizar *logout* e redefinição de senha. Para redefinição de senha faz-se o uso do método *sendPasswordResetEmail* da classe FirebaseAuth. Assim como no *e-mail* de verificação de novas contas, o servidor de *e-mail* para redefinição de senha é gerenciado pelo Firebase e ambas as mensagens são personalizadas no console do Firebase. Este recurso possibilita a formatação de uma mensagem mais amigável e intuitiva para os usuários, visto que o idioma padrão da ferramenta é na língua inglesa.

5.4.2 Provedor de localização

Obter a localização do dispositivo é uma tarefa diretamente ligada a fatores externos, principalmente relacionados com o ambiente onde ele encontra-se. Por exemplo, se há cobertura de sinal GPS, cobertura de sinal das redes de telefonia, existência de sinal Wi-Fi, entre outros aspectos abordados na seção 2.2.

Considerando tais situações, utilizou-se a classe Criteria para estabelecer o melhor provedor de localização disponível baseado nos parâmetros declarados em sua instância, como o nível de precisão da localização e o nível de consumo de bateria. Desta maneira foi possível obter o provedor de localização de forma dinâmica, sendo essa uma característica de extrema importância no contexto do trabalho desenvolvido, uma vez que essa abordagem permite alternar entre os provedores na medida em que o usuário transita por diferentes ambientes, sejam eles *indoor* ou *outdoor.*

Em relação aos parâmetros declarados em sua instância, definiu-se que o nível de precisão deve ser *fine*, com baixa margem de erro e sem necessidade de obter informações como a altitude e velocidade, o que levou a utilização do nível *ACCURACY_FINE.* Em relação ao consumo de bateria nada foi declarado, prevalecendo desta forma o *default NO_REQUIREMENT* que indica a não necessidade de requerimento especial para consumo de bateria.

A Figura 30 apresenta como a classe Criteria foi instanciada, parametrizada e passada para o método *requestLocationUpdates* que obtém a localização do dispositivo periodicamente:

Figura 30 – Utilização da classe Criteria

```
111         Criteria criteria = new Criteria();
112         criteria.setAccuracy(Criteria.ACCURACY_FINE);
113         mLocManager.requestLocationUpdates(Integer.parseInt(settingsMinTime),
114                 Integer.parseInt(settingsMinDistance),
115                 criteria,
116                 listener: this,
117                 Looper.myLooper());
```

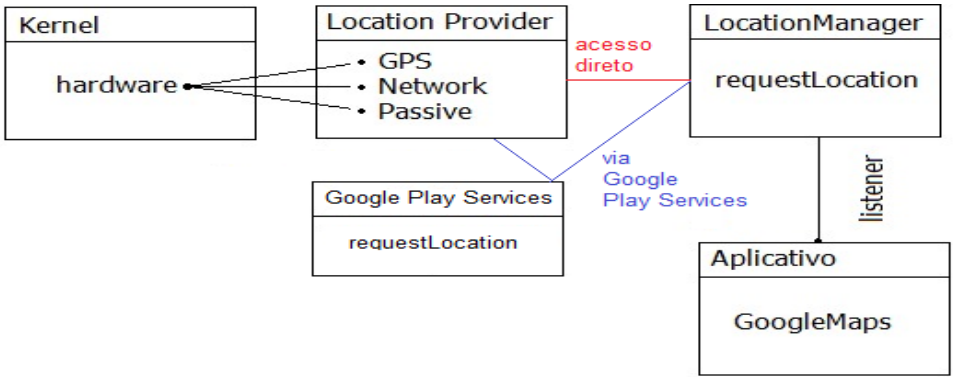
Fonte: Do autor (2020).

Como é possível observar na linha de código 111, a classe Criteria é instanciada e na linha seguinte o nível de precisão de localização é definido. Na linha 113 o método para obter atualizações periódicas sobre a localização é declarado, recebendo como parâmetros o tempo e distância mínima entre as atualizações, a instância da classe Criteria, o *listener* que receberá o *callback* das atualizações e um Looper.

Esta solução foi adotada nas duas rotinas do aplicativo que acessam a localização do dispositivo: quando o usuário está com o mapa aberto (seção 5.5.5) e quando o aplicativo está em segundo plano (seção 5.5.3). A única diferença entre elas é que na exibição do mapa utiliza-se o método *requestLocationUpdates* que recebe atualizações periódicas da posição, enquanto a rotina em segundo plano utiliza o método *requestSingleUpdate* que requer uma única atualização da posição para sincronizar com o Cloud Firestore.

Observou-se que na maioria dos casos o provedor de localização utilizado pela classe Criteria é o *fused.* Este provedor é gerenciado pelo Google Play Services e utiliza os diversos sensores do dispositivo para determinar sua posição. Isto é, quando a classe Criteria estabelece a utilização do provedor *fused,* ela não está fazendo um acesso direto aos sensores como GPS e A-GPS, por exemplo. Neste caso ela está fazendo uma requisição para o Google Play Services que, por sua vez fará os acessos aos sensores disponíveis naquele momento, funcionando como uma camada acima do nível de *hardware*. A Figura 31 ilustra este processo:

Figura 31 – Obtenção das coordenadas geográficas



Fonte: Do autor (2020).

É possível observar que sem a utilização do provedor *fused* há um acesso direto aos sensores do dispositivo, como o GPS, *network* e *passive*. Já com a utilização do provedor *fused* este acesso é intermediado pelo Google Play Services.

Além de utilizar a classe Criteria para definir o provedor dinamicamente, também foi feito outro teste visando implementar um comportamento semelhante. Este teste consistiu em criar uma lista de todos os provedores disponíveis antes de obter suas posições. Baseado nesta lista foi estabelecida uma hierarquia, tentando

obter primeiramente do provedor *network,* depois do *passive* e em últimos casos do GPS. Sempre que um provedor ficava disponível ou indisponível a lista era atualizada. Esse modelo também atendeu as expectativas e até oferece mais controles de acesso aos sensores, entretanto no contexto do aplicativo proposto a utilização da classe Criteria mostrou-se mais dinâmica e de simples utilização.

5.4.3 Funcionamento em segundo plano

Para obter a localização do dispositivo periodicamente e sincronizar essa informação em segundo plano com o Cloud Firestore, utilizou-se uma abordagem com AlarmManager e Broadcast Receiver. O AlarmManager é responsável por acionar a classe TrackerAlarm, que neste caso atua como Broadcast Receiver. Ao ser acionada, obtém a localização do dispositivo e sincroniza com o Cloud Firestore. É possível ver como o AlarmManager foi implementado através da Figura 32:

Figura 32 – Agendamento de rotinas *background*

```
294    public void scheduleAlarm(long seconds) {
295        Log.d( tag: "DEBUG",  msg: "Scheduling alarm");
296        alarm = (AlarmManager) getSystemService(ALARM_SERVICE);
297        Intent i = new Intent( packageContext: this, TrackerAlarm.class);
298        i.putExtra( name: "androidId", androidId);
299        i.putExtra( name: "settingStoreTrack", settingStoreTrack);
300        i.putExtra( name: "documentId", documentId);
301        pi = PendingIntent.getBroadcast( context: this,  requestCode: 1, i, PendingIntent.FLAG_UPDATE_CURRENT);
302        alarm.setInexactRepeating(AlarmManager.RTC_WAKEUP,  triggerAtMillis: 1000,  intervalMillis: seconds * 1000, pi);
303    }
```

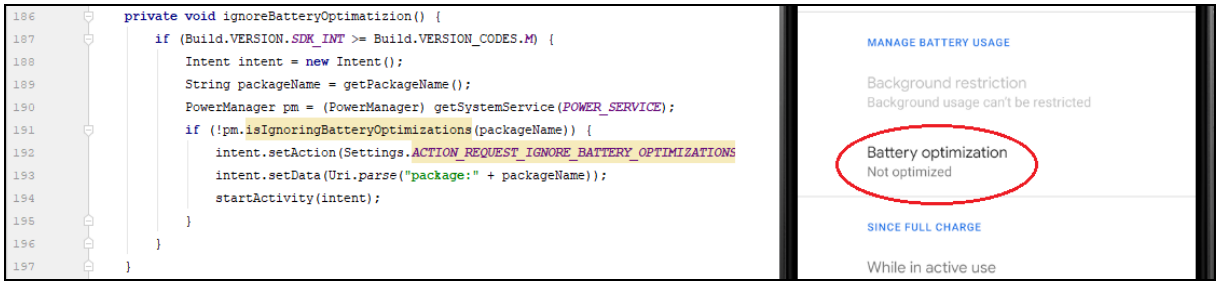
Fonte: Do Autor (2020).

A função *scheduleAlarm* foi criada para agendar a execução periódica do alarme, passando como parâmetro o intervalo de tempo desejado. Nesta função são instanciados um AlarmManager e uma *intent* para a classe TrackerAlarm. Como essa classe atua como um Broadcast Receiver, uma *pending intent* é instanciada passando a *intent* anterior como parâmetro. Em seguida o AlarmManager é agendado através do método *setInexactRepeating.*

Dois pontos importantes podem ser observados nesta etapa de agendamento. O primeiro deles é a necessidade de remover o aplicativo das configurações de otimização de bateria em dispositivos com versão Android 7.0 ou superior. Do

contrário, o alarme não é executado se o dispositivo está bloqueado ou quando estiver no modo de economia de bateria. Esta situação ocorre em função de alguns recursos do próprio Android que visam preservar a bateria, como o DOZE MODE. O código utilizado para realizar a requisição de não otimização de bateria pode ser observada na Figura 33.

Figura 33 – Requisição para não otimização de bateria



Fonte: 1Do Autor (2020)

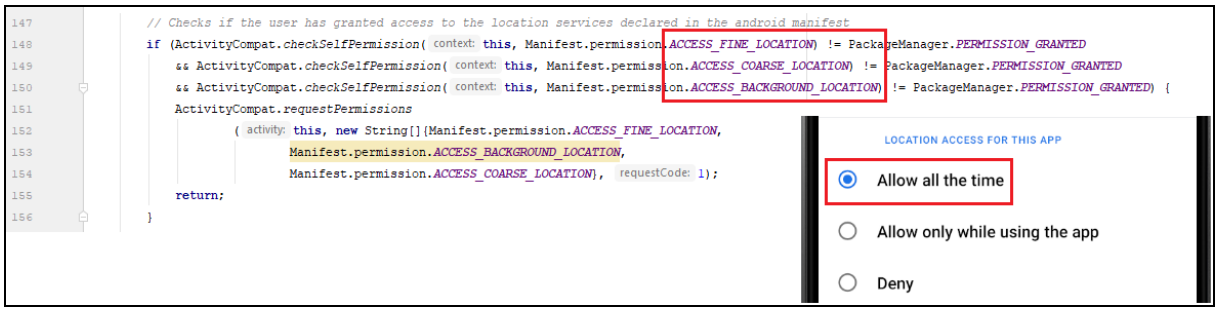
O segundo ponto é que a utilização do método *setInexactRepeating* se mostrou mais eficiente do que o método *setRepeating* em relação ao consumo de bateria. O motivo é que desta maneira o Android sincroniza alarmes recorrentes para executá-los de uma só vez, evitando que o dispositivo tenha que ser acionado mais vezes. Como a precisão no tempo de execução do alarme não é relevante no contexto do aplicativo desenvolvido, a utilização deste método mostrou-se mais adequado.

Quando a classe TrackerAlarm é acionada, o método *onReceive* inicia as rotinas para obter a localização do dispositivo. Para determinar o provedor de localização utiliza-se a classe Criteria, cujo funcionamento é descrito na seção 5.5.2. Nos primeiros testes a localização era buscada no *cache* do dispositivo, através do método *getLastKnownLocation,* porém esse método mostrou-se inadequado porque muitas vezes a localização não era atualizada há muito tempo ou não estava mais em *cache*, retornando nulo. Assim sendo, utilizou-se o método *requestSingleUpdate* que força o dispositivo a atualizar sua localização.

Depois que a localização foi obtida, a latitude e longitude são convertidas em endereços humanos (bairro e cidade) através da API Geocoder e sincronizadas com o Cloud Firestore. As informações também são armazenadas na coleção *userHistory* caso o usuário esteja configurado para armazená-las.

O correto funcionamento destas rotinas requer que o usuário permita acesso a localização em segundo plano, como mostra a Figura 34. Do contrário, o aplicativo não tem permissão para requisitar a localização do dispositivo quando ele não estiver sendo executado, o que inviabilizaria totalmente sua proposta.

Figura 34 – Requisição para acesso de localização em segundo plano



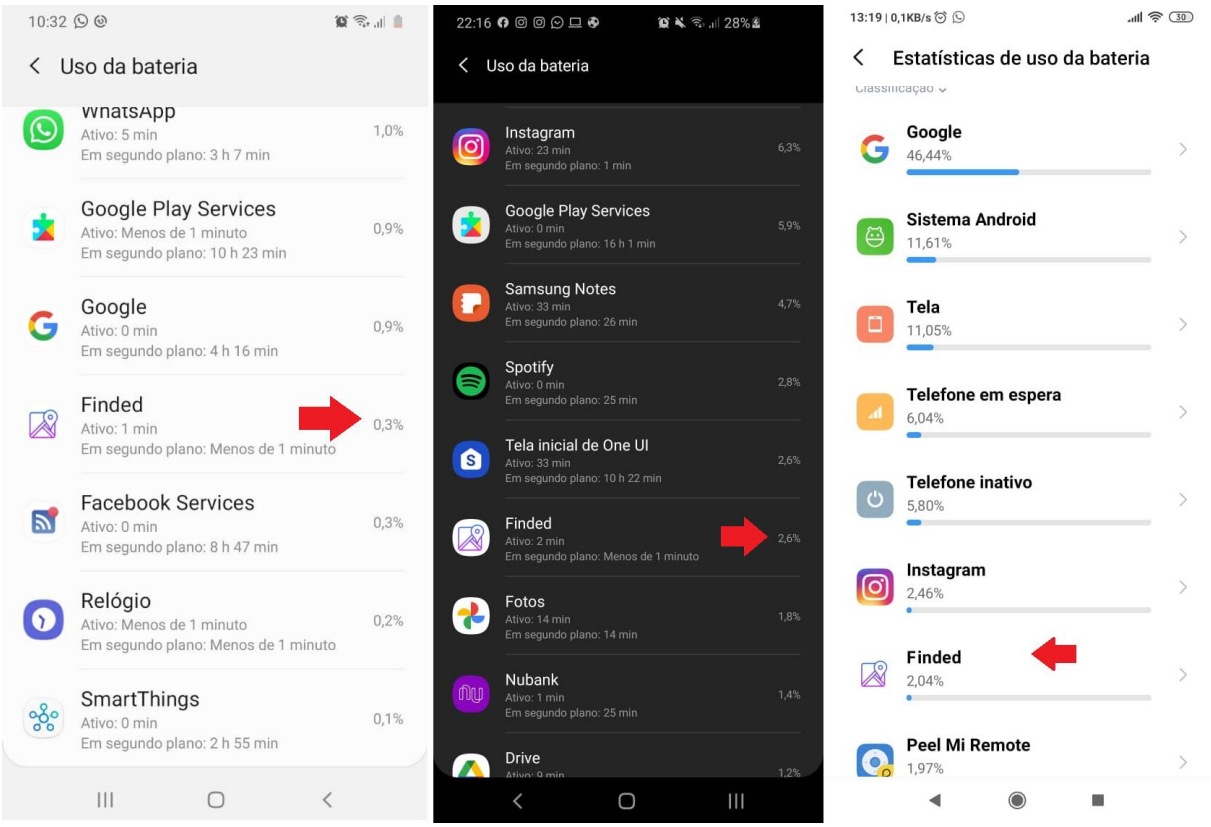
Fonte: Do Autor (2020).

Uma das preocupações ao utilizar processamento em segundo plano refere-se ao consumo de bateria do dispositivo. Um aplicativo que consome muita bateria pode causar uma experiência desagradável ao usuário e prejudicar o desempenho do aparelho. Assim sendo, foram realizados testes em diferentes dispositivos, onde agendou-se a execução de alarmes a cada dois minutos durante um período de 24 horas. Durante este tempo o aplicativo também foi executado em primeiro plano algumas vezes.

O resultado dos testes pode ser visto na Figura 35, onde concluiu-se que o percentual de consumo de bateria é satisfatório, principalmente em relação a outros aplicativos como o Spotify, Whatsapp e Instagram. No dispositivo de marca Xiaomi, o consumo de bateria atingiu apenas 0,3%, enquanto no Samsung e Motorola atingiu 2,6% e 2,04%, respectivamente.

Outras abordagens também foram testadas para obtenção e sincronização de dados em segundo plano. O objetivo era utilizar uma solução que atendesse aos requisitos e minimizasse o uso de bateria do dispositivo. Neste sentido, a API JobScheduler também foi eficiente, ao contrário da utilização de Services que, por terem uma *thread* em execução o tempo inteiro, acaba consumindo altos níveis de bateria, chegando a 28% em um dos testes realizados no dispositivo Samsung.

Figura 35 – Uso de bateria em segundo plano com AlarmManager



Fonte: Do Autor (2020).

5.4.4 Persistência dos dados e sincronização com a nuvem

Uma das premissas do aplicativo é sincronizar informações com a nuvem para que seja possível compartilhá-las com outros usuários através de uma rede de contatos. Entretanto, considerando que nem sempre haverá conexão disponível com a *internet*, é preciso armazenar os dados localmente até que a conexão seja reestabelecida.

Para lidar com este tipo de situação utilizou-se o Cloud Firestore que pode ser configurado para armazenar os dados em *cache* antes de sincronizá-los, atendendo ao requisito do aplicativo. Para elucidar como foi feita a utilização dessa ferramenta no aplicativo desenvolvido, pode-se separar as ações em dois tipos: inserção e consulta. Os parágrafos a seguir explicam como estas duas ações ocorrem no aplicativo Finded.

A inserção de dados sempre é feita localmente e fica armazenada na memória *cache* do aplicativo. Como foi dito anteriormente, a sincronização com a nuvem dependerá de conexão com a *internet.* É o que acontece ao incluir um novo registro no histórico de localização do usuário (método *add*) ou atualizar as informações de um usuário já existente (método *update*) como mostra a Figura 36.

Figura 36 – Alteração e inclusão de documentos

Fonte: Do autor (2020).

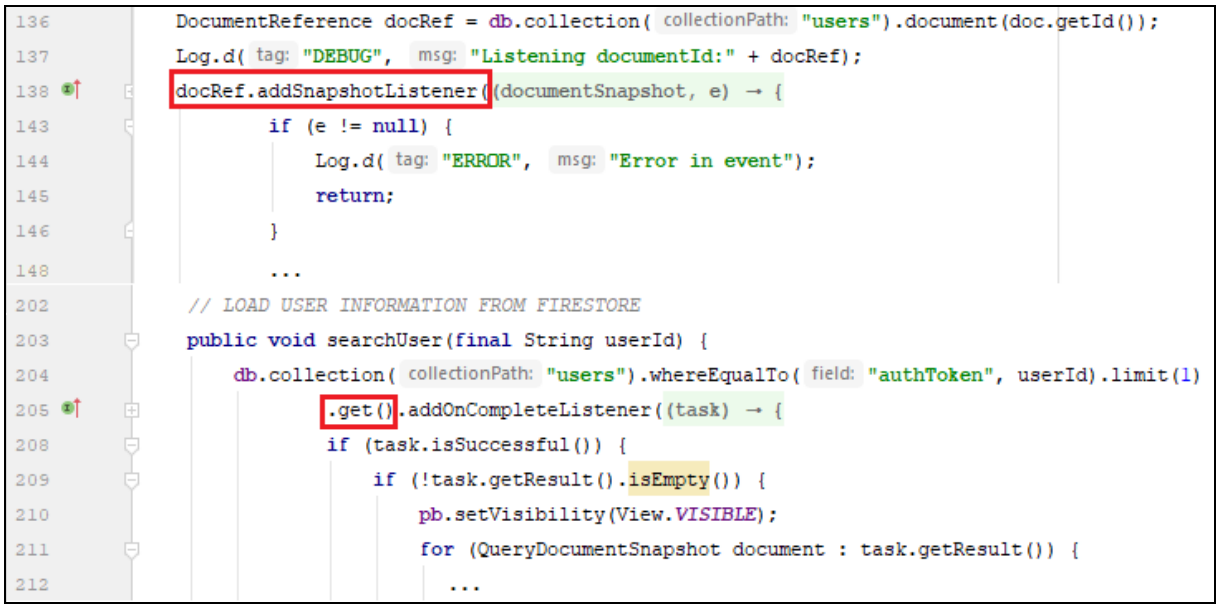
Neste exemplo, um documento teve campos atualizados (*update*) e outro documento foi criado (*add*). Ambos são automaticamente sinalizados como pendentes de sincronização com o *backend* para que desta forma o Cloud Firestore tenha controle sobre as pendências. Este tipo de comportamento é comum em várias rotinas do aplicativo, como na criação de grupos, atualização das informações do usuário, envio de mensagens no *chat etc.*

Em relação a consulta de dados, foi adotada uma lógica que sempre tenta consultar os dados *online,* isto é, na nuvem. Caso não seja possível conectar-se com os servidores do Cloud Firestore, então o aplicativo trabalha com os dados que já foram sincronizados anteriormente. Isso permite que o usuário veja os grupos que ele faz parte, o histórico de mensagens do *chat* e as últimas localizações da sua

rede de contatos, mesmo que não tenha conexão com a *internet*. Logicamente, não é possível receber novas atualizações até que a conexão seja reestabelecida.

É o que acontece ao carregar as informações básicas do usuário conectado (método *get*) ou ligar um *Snapshot Listener* na coleção de membros do grupo (método *addSnapshotListener*). Sua implementação pode ser vista na Figura 37:

Figura 37 – Consulta de documentos



Fonte: Do autor (2020).

Neste exemplo, se não houver conexão com a *internet*, as informações em *cache* serão carregadas. O mesmo ocorrerá no caso do *Snapshot Listener*, porém os documentos não irão receber novas atualizações até reestabelecer a conexão.

Este é o comportamento básico adotado para inserir, atualizar e consultar documentos do Cloud Firestore. Para avançar em seu entendimento, pode-se continuar observando a Figura 37. Como mostra a linha de código 205, após a declaração do método *get*, outro método denominado *addOnCompleteListener* é declarado. Este método recebe como parâmetro uma *Task* do tipo *QuerySnapshot* que, ao ser finalizada aciona o método *onComplete* que atua como *callback*. Dentro deste método a lógica dos dados é implementada, como por exemplo, exibi-los na tela. Comportamento semelhante acontece com o método *addOnSnapshotListener* na linha de código 138, que recebe como parâmetro um objeto do tipo *EventListener*. Na medida em que os *listeners* são acionados, o método *onEvent*

aplica a lógica sobre os dados recebidos, como redesenhar a posição de um usuário no mapa ou mostrar uma mensagem no *chat.*

Outra situação existente em algumas rotinas é a execução de consultas compostas, passando condições e referenciando outros documentos. Em outras rotinas é necessário realizar uma primeira consulta para armazenar suas referências em memória, para que em seguida uma segunda consulta seja executada baseada nas referências retornadas pela primeira. É o que acontece quando o usuário consulta os grupos que ele faz parte, onde uma *query* é executada para realizar a consulta. Para cada um dos grupos retornados, uma nova *query* é executada para consultar os detalhes daquele grupo, como a lista de membros, nome do grupo, descrição e a data de criação. A Figura 38 apresenta um log do aplicativo que auxilia no entendimento do processo:

Figura 38 – Exemplo *log* de consulta

```
1  D/DEBUG: Loading groups with get() method.
2  D/DEBUG: Task 1 sucessfull, returned groups: KVOI-6626, IQBN-4666, CKHL-3365, CMCK-4268
3  D/DEBUG: Starting to consult members of each group
4  D/DEBUG: Task 1 Loanding group's members.
5  D/DEBUG: Task 2 Loanding group's members.
6  D/DEBUG: Task 3 Loanding group's members.
7  D/DEBUG: Task 4 Loanding group's members.
8  D/DEBUG: Task 2 sucessfull. Group [ KVOI-6626 ], members [ Matheus, Leandro, Alceu, Bete]
9  D/DEBUG: Task 3 sucessfull. Group [ IQBN-4666 ], members [ Matheus, Ana ]
10 D/DEBUG: Task 1 sucessfull. Group [ CMCK-4268 ], members [ Matheus ]
11 D/DEBUG: Task 4 sucessfull. Group [ CKHL-3365 ], members [ Walter, Gustavo, Matheus ]
```

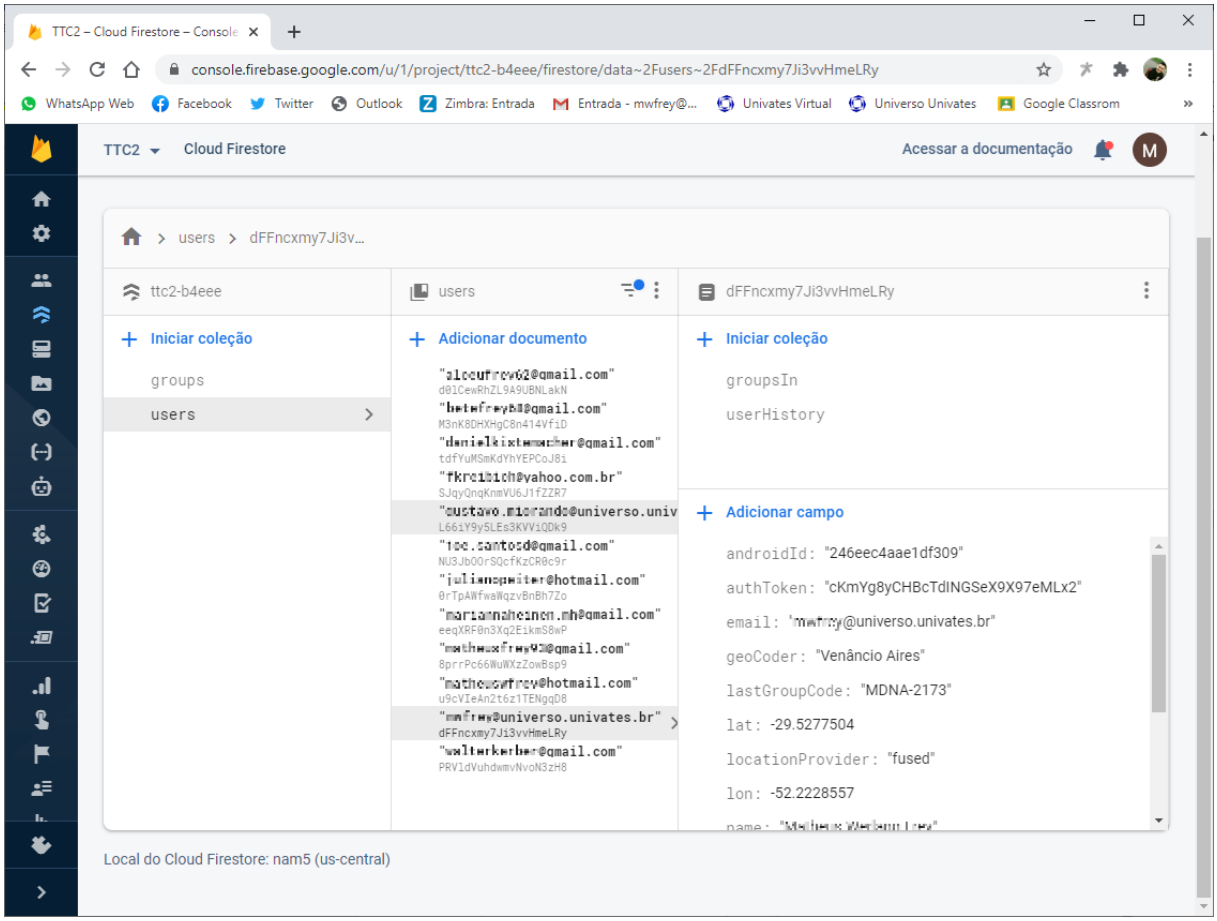
Fonte: Do autor (2020).

Observa-se na linha de código 2 que na primeira *task* retornou a identificação dos 4 grupos que o usuário faz parte. Na sequência um laço de repetição criou quatro *tasks* para retornar os membros de cada um destes grupos, conforme indicam as linhas de código 4, 5, 6 e 7. Por característica, estas *tasks* são assíncronas e na medida em que suas *threads* são finalizadas, o *callback* é acionado. Por este motivo elas não foram finalizadas necessariamente na ordem que foram lançadas, como mostram as linhas de código 8, 9, 10 e 11.

Este tipo de consulta está diretamente relacionada com a estruturação da coleção de dados. Todas essas informações poderiam ser gravadas em ambas as coleções para evitar a realização de consultas compostas, mas por outro lado este tipo de modelo resultaria em redundância de dados porque estariam gravados em diferentes coleções. Por este motivo, optou-se pelo modelo descrito.

Toda a base de dados do Cloud Firestore pode ser facilmente acessada via console *web,* cuja aparência é ilustrada na Figura 39.

Figura 39 – Console Firebase



Fonte: Do autor (2020).

Através deste console é possível monitorar o uso do Cloud Firestore, realizar diversos tipos de manutenções na base de dados, criar índices personalizados de consulta e alterar as regras de segurança do Cloud Firestore. Outro característica é que a base de dados pode ser facilmente compartilhada com outros projetos caso o desenvolvedor julgue adequado.

Para realizar a configuração do Cloud Firestore foi necessário habilitá-lo no console do Firebase e em seguida adicionar a dependência *firebase-firestore:17.1.2* no aplicativo. Entretanto, ocorreu o seguinte erro ao tentar executar o aplicativo: "*Cannot fit requested classes in a single dex file (# methods: 85256 < 65536)".*

Identificou-se na documentação do Firebase que este erro ocorre em versões 5.0 ou anteriores, onde a execução dos *bytecodes* é realizada pelo método JIT (seção 2.3.1). Essas versões limitam os aplicativos a utilizarem um único arquivo de *bytecode Dalvik executable* (DEX), e neste caso, o número máximo de referências (65536) foi excedido.

Contudo, é possível ligar a configuração *Multidex* que permite gerenciar mais do que um arquivo DEX pelo aplicativo. Para ligar a configuração é necessário acessar o *build.gradle* do aplicativo, incluir a variável *defaultConfig* como *true* e adicionar a dependência *Multidex 1.0.3*.

Como adotou-se a lógica de armazenar e manter os documentos em *cache*, a utilização de armazenamento do dispositivo foi uma das preocupações durante o desenvolvimento, dado que alguns aparelhos não possuem grande capacidade de armazenamento. Assim sendo, realizou-se um teste para observar a utilização deste recurso, que consistiu em armazenar o histórico das rotas a cada dois minutos, entrar em dez grupos e executar um algoritmo para geração de mensagens em massa no *chat.* Através deste teste observou-se que os dispositivos utilizaram em média 33,97 MB de armazenamento, o que classifica o resultado como satisfatório.

Considerando situações eventuais onde o armazenamento venha a ser um problema, uma alternativa é executar uma limpeza de *cache* nas configurações do Android, afinal eles podem ser baixados novamente caso voltem a ser necessários. Outra maneira seria a implementação de uma lógica para que o próprio aplicativo execute uma limpeza de documentos velhos.

Uma segunda abordagem utilizando SQLite para persistência de dados locais e PostgreSQL para armazenamento *online* de dados também foi testada durante o desenvolvimento do trabalho. Embora essa abordagem atenda aos requisitos do aplicativo proposto, este formato apresenta algumas desvantagens em relação ao Cloud Firestore para o aplicativo proposto.

Primeiramente o tempo de desenvolvimento é consideravelmente maior, visto que a integração entre os sistemas SQLite e PostgreSQL deve ser implementada, bem como toda a lógica de sincronização via *socket*. Outra desvantagem é disponibilizar uma infraestrutura que garanta o alto desempenho e disponibilidade

dos serviços, visto que com o Cloud Firestore estes serviços são garantidos pelo Google.

Contudo a maior desvantagem observada neste formato é a necessidade de implementar rotinas de execução em segundo plano para efetuar a sincronização entre SQLite e PostgreSQL. Por exemplo, ao utilizar intervalos de tempo muito grandes para sincronização, a sensação de tempo real é prejudicada. Já com intervalos muito pequenos, a bateria do dispositivo é sobrecarregada. Neste sentido, os *listeners* do Cloud Firestore se mostram muito mais adequados porque além do processamento ser executado fora da *main thread,* eles recebem notificações somente quando há alguma modificação para ser sincronizada, ou seja, a sincronização ocorre de fato em tempo real.

Para a realização dos testes utilizando SQLite e PostgreSQL, primeiramente o SQLite foi configurado no aplicativo para armazenar os dados *offline*. Em seguida uma biblioteca JDBC foi importada para fazer a comunicação com o PostgreSQL, que foi hospedado em um servidor com sistema operacional Ubuntu 19.04, localizado na Univates e com endereço de IP externo para conexões remotas. Ao realizar uma consulta na nuvem um objeto ResultSet recebe a conexão com o banco de dados e a *query* de consulta SQL é executada. Essa tarefa é realizada em segundo plano através de um AlarmManager e o resultado da consulta é inserido no SQLite.

5.4.5 Mapa

A Activity Maps é responsável por projetar o mapa na tela, realizar interações e indicar a posição do usuário e dos membros do grupo selecionado. Para sua construção, a primeira etapa foi prepará-la para receber a projeção do mapa, onde foi criado um *Fragment* e atrelado a classe *suportMapFragment*.

A etapa seguinte tratou de realizar a integração com o Google Maps API, sendo necessário importar as dependências do *services-maps:17.0.0*, gerar a API Key para autenticar requisições e informá-la no *AndroidManifest.xml*.

Em seguida foram feitas implementações para obter a posição do dispositivo através do método *requestLocationUpdates* da classe *Location Listener* e desenhar essa posição no mapa através da classe *Markers*. O método recebe dois parâmetros para especificar a distância e tempo mínimo entre as atualizações e desta forma o *callback* somente será acionado se atender as condições declaradas, evitando o processamento desnecessário. No método *onLocationChanged*, isto é, no *callback* do *Location Listener,* foi criada uma lógica para redesenhar a posição do *Marker* no mapa conforme mostra a Figura 40.

Figura 40 – Função para projetar a posição do usuário atual no mapa

```
200     @Override
201     public void onLocationChanged(Location location) {
202         posCurrentUser = new LatLng(location.getLatitude(), location.getLongitude());
203         twTittle.setText("\n       " + groupName + "       \n");
204         if (markerSelfUser == null) {
205             markerSelfUser = mMap.addMarker(new MarkerOptions().position(posCurrentUser));
206             markerSelfUser.setIcon(bitmapDescriptorFromVector(getApplicationContext(), R.drawable.ic_baseline_adjust_24));
207             mMap.moveCamera(CameraUpdateFactory.newLatLng(posCurrentUser));
208             mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(posCurrentUser, v: 8.0f));
209             recenterUser.setVisibility(View.VISIBLE);
210         } else {
211             markerSelfUser.setPosition(posCurrentUser);
212         }
213         markerSelfUser.setTitle("Você");
214     }
```

Fonte: Do autor (2020).

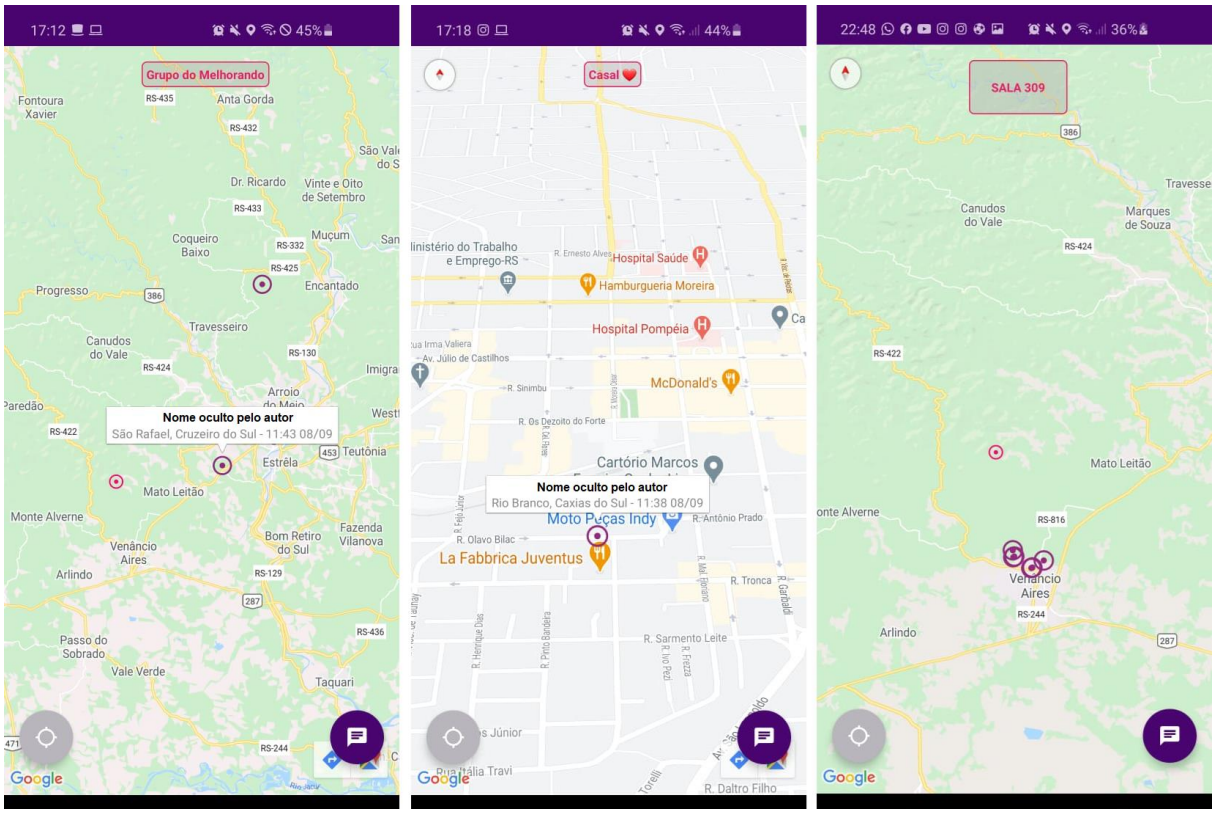
Neste exemplo observa-se que a posição do usuário é armazenada em uma variável e em seguida uma validação é realizada para saber se o *Marker* já está desenhado no mapa (linha de código 204). Caso não esteja, a variável *markerSelfUser* é instanciada, o *Marker* é desenhado e o foco da tela é movido para a posição do usuário. Na prática essa condição é atendida somente quando o usuário abre o aplicativo, pois depois a execução cai na cláusula *else* e a posição do *Marker* apenas é redesenhada, sem a necessidade de criá-lo novamente.

Por último, foi criada uma lógica para obter a posição dos outros membros do grupo e projetá-las no mapa, lógica esta que é regida pelo *Snapshot Listener* do Cloud Firestore. Quando o usuário inicia a Activity Maps, além de todas as rotinas descritas nesta seção, o aplicativo também realiza uma consulta no Cloud Firestore para ver quem são os membros do grupo conectado e os projeta na tela. Em seguida um laço de repetição cria um *Snapshot Listener* para cada um destes membros e na medida em que eles sincronizam sua posição com o Cloud Firestore

(seção 5.5.3) o *callback* é acionado e a sua posição é redesenhada no mapa. Essa lógica é muito parecida com a implementação do *chat* que será abordada na seção 5.5.5. O tempo de resposta do *Snapshot Listener* ocorre em poucos milissegundos, portanto, se um membro enviar sua posição para o Cloud Firestore, em pouco tempo todos os outros serão notificados para que a posição seja atualizada no mapa.

O resultado de todas estas implementações pode ser observado na Figura 41 onde o *Marker* rosa indica a posição do usuário conectado e os *Markers* roxos indicam a posição dos outros membros do grupo. Ao clicar sobre o membro é possível ver mais detalhes sobre ele, como a data/hora da última sincronização realizada e o bairro/cidade que ele se encontra, tarefa realizada pela classe Geocoder que possibilita converter coordenadas geográficas em endereços humanamente compreensíveis.

Figura 41 – Exemplos mapa



Fonte: Do autor (2020).

Outras funcionalidades adicionais também foram implementas, como o botão para centralizar a posição atual do usuário, um botão para abrir o *chat* do grupo e as

funções *moveCamera* e *animateTo* para mover a câmera automaticamente quando o usuário está se movendo pelo mapa.

Para melhor experiência do usuário, durante a exibição do mapa nenhum processamento pesado ou com alta latência é executado na *main thread*. O motivo é que este tipo de comportamento pode atrapalhar a interação com o mapa e causar a sensação de travamento.

Por exemplo, quando o *Marker* é redesenhado, a *main thread* não aciona a classe Geocoder para converter as coordenadas em endereços humanos, visto que este procedimento requer conexão com a *internet* e exige um certo tempo de resposta. Para lidar com este tipo de situação procurou-se utilizar as *Tasks* que são executadas em segundo plano e notificam a *main thread* ao serem concluídas. É o que acontece com as sincronizações dos *Snapshot Listeners* e a utilização do método *onLocationChanged*, por exemplo.

Até então, a maioria dos testes eram feitos na máquina virtual do Android Studio, entretanto, para testar a interação com o mapa e a atualização da posição em tempo real foi necessário utilizar aparelhos reais durante a locomoção. Alguns destes testes foram realizados em Linha Cecília, interior de Venâncio Aires/RS e outros por voluntários que residem na zona urbana de Venâncio Aires/RS, Caxias do Sul/RS, Cruzeiro/RS e Lajeado/RS.

5.4.6 Chat

A implementação do *chat* pode ser dividida em duas etapas, sendo a primeira delas relacionadas ao seu comportamento e a segunda em relação ao seu *layout.* Os parágrafos a seguir trazem mais detalhes sobre seu desenvolvimento.

Em relação ao comportamento, a primeira etapa consistiu em buscar as mensagens do grupo no Cloud Firestore e armazená-las em ArrayLists. Em seguida um *listener* é ativado na coleção *messages* daquele grupo afim de detectar novas mensagens. Sempre que o *callback* do *listener* for acionado, uma rotina verifica se o evento é uma inclusão, alteração ou exclusão de documento. Tratando-se de

inclusão, a mensagem é incluída nos ArrayLists. Este comportamento é ativado por uma função denominada *messagesListener*, a qual é mostrada na Figura 42.

Figura 42 – Implementação dos *chat* com utilização de *listeners*

```
100     public void messagesListener() {
101         db.collection( collectionPath: "groups") CollectionReference
102                 .document(groupId) DocumentReference
103                 .collection( collectionPath: "messages").orderBy("at") Query
104                 .addSnapshotListener(MetadataChanges.INCLUDE, new EventListener<QuerySnapshot>() {
105             @Override
106             public void onEvent(@Nullable QuerySnapshot queryDocumentSnapshots, @Nullable FirebaseFirestoreException e) {
107                 if (e != null) {
108                     Log.d( tag: "Listener error",  msg: ":" + e);
109                     return;
110                 }
111                 for (DocumentChange dc : queryDocumentSnapshots.getDocumentChanges()) {
112                     if (dc.getType() == DocumentChange.Type.ADDED) {
113                         if (!queryDocumentSnapshots.getMetadata().hasPendingWrites()) {
114                             // Lógica para exibição da mensagem
115                         }
116                     }
117                 }
118                 setupAdapter();
119             }
120         });
121     }
```

Fonte: Do autor (2020).

Observa-se que nas linhas de código 101, 102 e 103 a referência é montada para execução da consulta e na linha seguinte o método *addSnapshotListener* é declarado. O *callback* está na linha 106 e na sequência uma cláusula *if* verifica se existe algum *exception* no evento retornado e, caso não tenha, um laço de repetição *for* é executado sobre os documentos retornados.

Dentro deste laço existe uma validação para identificar se o documento é inclusão de documento (linha de código 112) pelo fato de que não foi implementada lógica para exclusão e alteração de mensagens. Na sequência, outra validação é realizada (linha de código 113) para descartar mensagens escritas pelo usuário conectado, visto que as alterações locais já estão sincronizadas no dispositivo e portanto apenas são gravadas diretamente nos ArrayLists. Já na linha de código 114 é implementada a lógica para exibição das mensagens recebidas e, em função do extenso código, ela foi cortada da imagem.

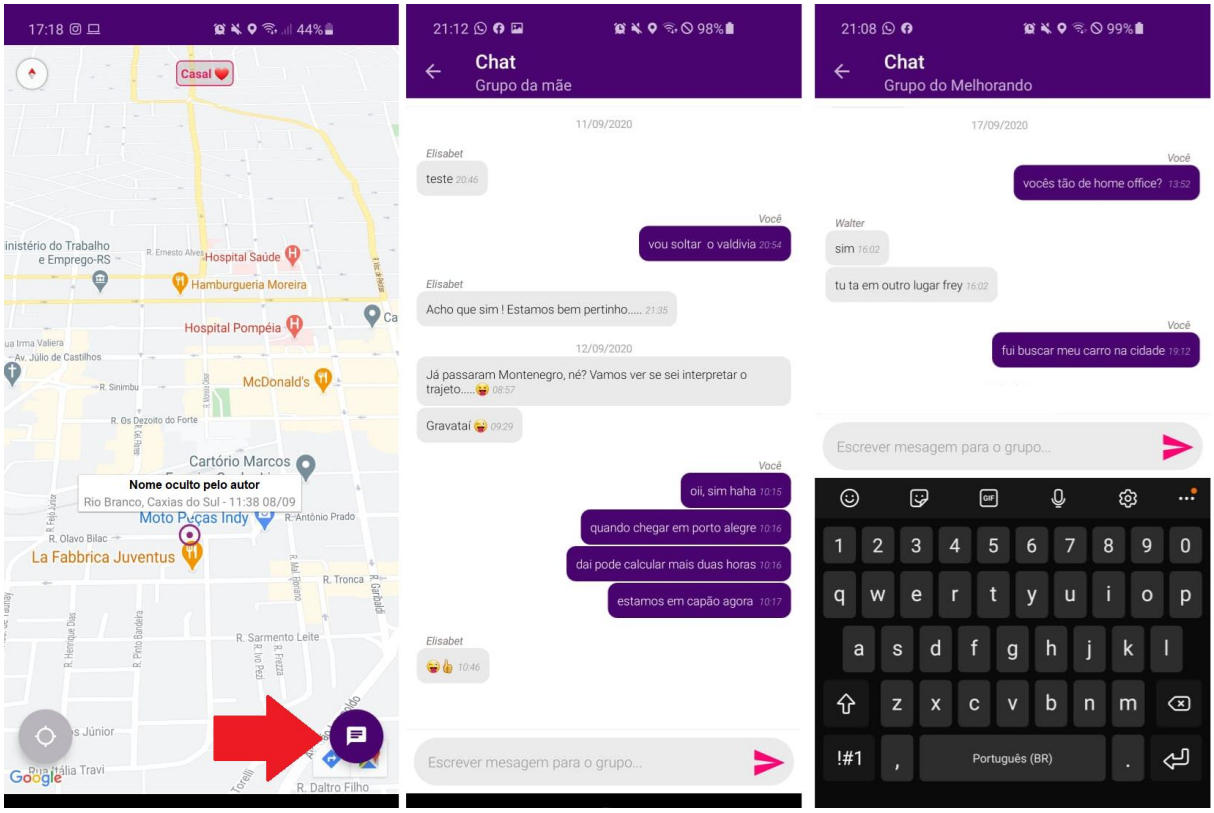
Em relação ao *layout,* foi feita a utilização das classes Adapter e Recycler View. Neste modelo, as mensagens são incluídas nos ArrayLists e enviados para outra classe que estende as classes RecyclerView.Adapter. Essa classe, por sua

vez, é encarregada de percorrer os *arrays* e exibir as mensagens através de *layouts* do tipo Recycler View, permitindo desta forma a rolagem das mensagens na tela.

O *layout* segue os padrões já conhecidos do Android, onde as mensagens enviadas são alinhadas à direita e as mensagens recebidas à esquerda. Para isso, dois *layouts* do tipo Recycler Row foram criados, um para mensagens recebidas e outro para mensagens enviadas. Ao receber novos elementos nos ArrayLists, é feita uma verificação no método *onCreateViewHolder* para determinar qual dos dois *layouts* o índice irá seguir. Uma lógica também foi desenvolvida para agrupar as mensagens em sequência de um mesmo remetente visando ter um melhor aproveitamento da tela.

Para escrever as mensagens foi criado um componente do tipo EditText e para enviá-las, um componente do tipo Button, cuja função é capturar o evento de clique e adicionar os dados da mensagem nos respectivos. O resultado pode ser observado na Figura 43, onde a primeira imagem exibe o ícone para acesso ao *chat* e as outras duas seu o *layout*.

Figura 43 – Layout do *chat*



Fonte: Do autor (2020).

5.4.7 Notificações *push*

Sempre que um usuário escrever uma mensagem no *chat* e algum dos membros não estiver com o aplicativo sendo executado em primeiro plano, uma notificação *push* é exibida aos destinatários. Para que isso seja possível utilizou-se três recursos do Firebase em conjunto: Cloud Firestore, Cloud Functions e Cloud Messaging.

A lógica para recebimento, processamento e exibição das notificações é executada no dispositivo cujo aplicativo Finded está instalado, isto é, no *client-side*. Já a lógica para geração das notificações é executada no *backend,* que por sua vez é hospedado no Cloud Functions, ou seja, no *server-side*.

No *client-side* criou-se uma classe que estende *FirebaseMessagingService* e implementa os métodos responsáveis por registrar o *token* de identificação no Cloud Messaging (*onNewToken)* e exibir as notificações ao receber novas mensagens (*onMessageReceived)*. Ou seja, o método *onNewToken* tem a função de registrar um *token* de identificação para cada usuário quando o aplicativo é executado pela primeira vez. O método *onMessageReceived* é responsável por escutar novas mensagens e exibir as notificações *push*. O próprio Firebase Cloud Messaging exibe as notificações automaticamente desde que o aplicativo não esteja sendo executado em primeiro plano e que a mensagem tenha o *token notification.*

Para identificar os usuários que devem ser notificados sobre as mensagens no grupo utilizou-se o conceito de tópicos. Ou seja, sempre que um usuário entrar em um novo grupo, ele será automaticamente inscrito naquele tópico e passará a receber notificações quando novos documentos forem incluídos na coleção. Essa inscrição é realizada método *subscribeToTopic(String s)* do Cloud Messaging.

A lógica de tópicos foi adotada porque desta maneira é mais fácil de gerenciar os destinatários das notificações. Do contrário seria necessário armazenar o *token* de cada usuário no Cloud Firestore e enviar as notificações individualmente.

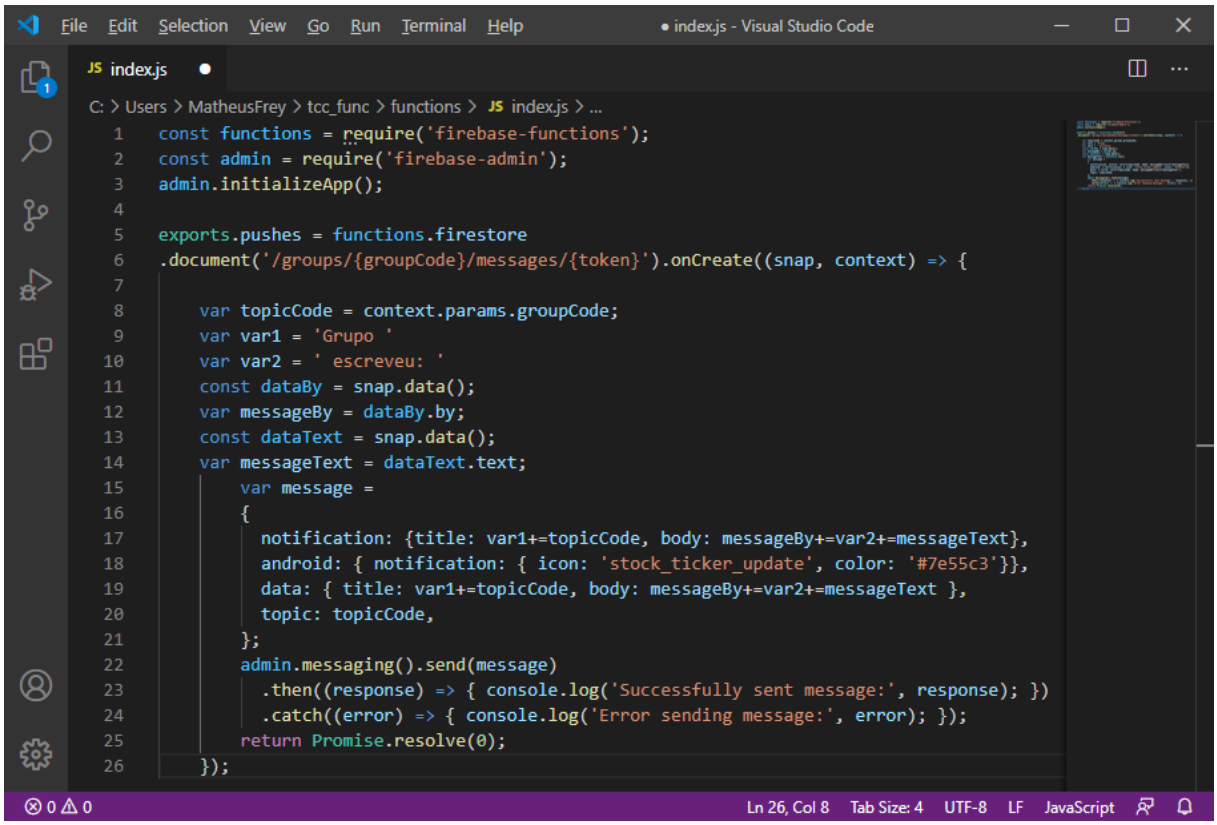
A partir deste momento já era possível gerar notificações através do console do Firebase e direcioná-las para determinados dispositivos, entretanto para o correto

desenvolvimento deste trabalho é necessário que as notificações sejam disparadas automaticamente quando mensagens forem escritas pelos usuários.

Assim sendo, o próximo passo consistiu em implementar o *backend* para gerar as notificações através de uma *trigger* que foi hospedada no Cloud Functions. Essa *trigger* monitora eventos na coleção *messages* do Cloud Firestore e sua função é enviar dados ao Cloud Messaging quando novos documentos forem incluídos na coleção.

Para elaboração da *trigger* foi criado um projeto no Node.js e vinculado ao projeto do Firebase. No arquivo *index.js* toda a lógica para captura de eventos e disparo de notificações foi implementada conforme mostra a Figura 44. Feito isso, um *deploy* tratou de hospedar o projeto no Cloud Functions e a partir deste momento os destinatários passaram a receber notificações para mensagens enviadas nos *chats* que eles estavam inscritos.

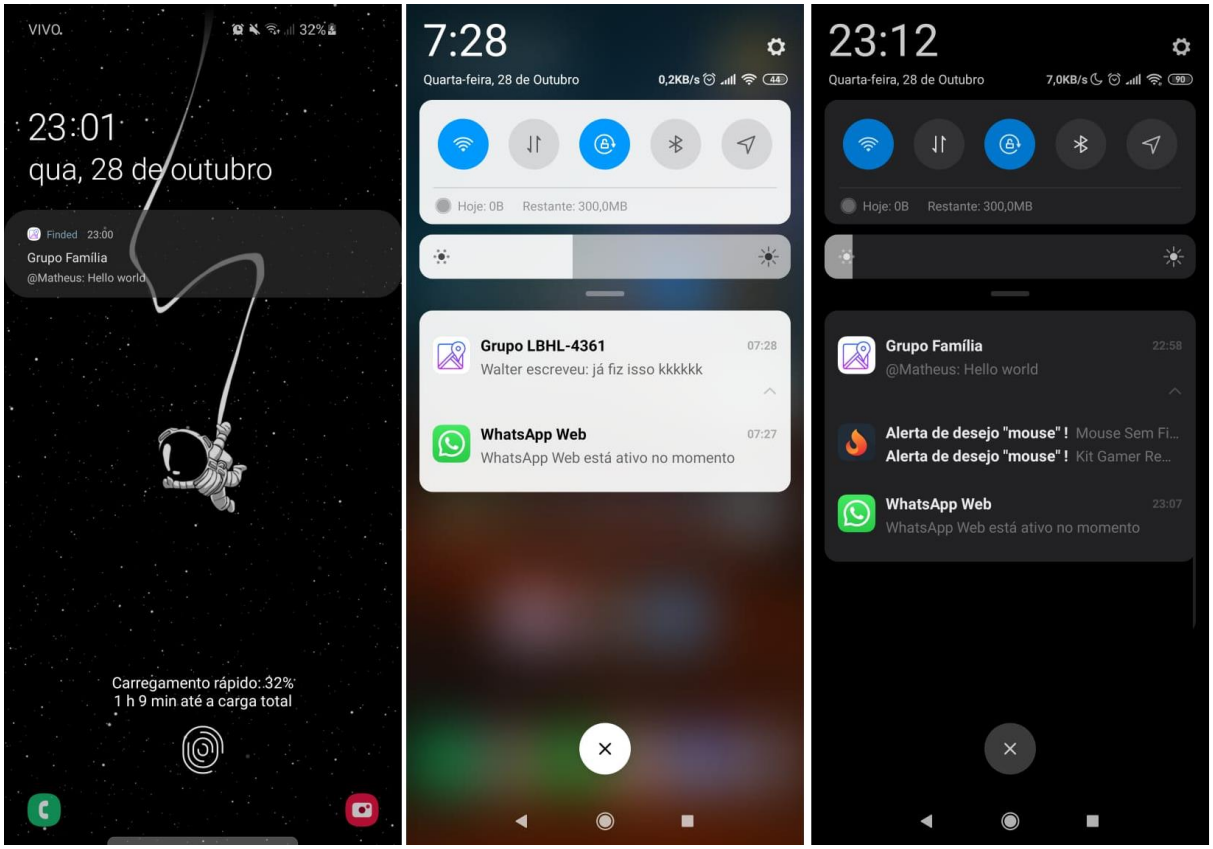
Figura 44 – Classe *index.js* para captura de eventos e geração de notificações



Fonte: Do autor (2020).

Observando a Figura 44, nota-se que nas linhas 1 e 2 as funções do Firebase e SDK Admin são requeridas, enquanto na linha 4 a função *“pushes”* é declarada e exportada. Na linha 5 a coleção é especificada com uso de caracteres curinga para identificar o grupo da mensagem e *token* do documento. Na linha 13 o JSON da notificação é estruturado e na linha 20 ele é passado como parâmetro para a função *Messaging().send(message)* do SDK Admin, que trata de enviá-la aos usuários. A Figura 45 traz um exemplo de como as notificações são exibidas nos dispositivos.

Figura 45 – Exemplo notificação



Fonte: Do autor (2020).

Apesar de que só tenham sido implementadas funções para monitorar a inclusão de novos documentos e gerar notificações *push*, a estrutura do Cloud Functions permite manipular outros eventos diretamente no Firebase, possibilitando que inúmeras funcionalidades sejam criadas no *backend* do aplicativo Finded, como pesquisa de lugares nas imediações, condições do trânsito, envio de convites para participação de grupos, e muitas outras facilidades.

5.4.8 Configurações do aplicativo

Uma tela de configurações foi criada para definir alguns parâmetros de execução. Ela pode ser acessada através do menu *Settings* existente na *ActionBar* da Activity principal. É possível fazer as seguintes definições:

- Gerar histórico de localização (*default false)*: se ativado, armazena o histórico de localização na coleção *userHistory* do usuário conectado. Embora o histórico possa ser armazenado, ainda não é possível acessá-lo pelo aplicativo;

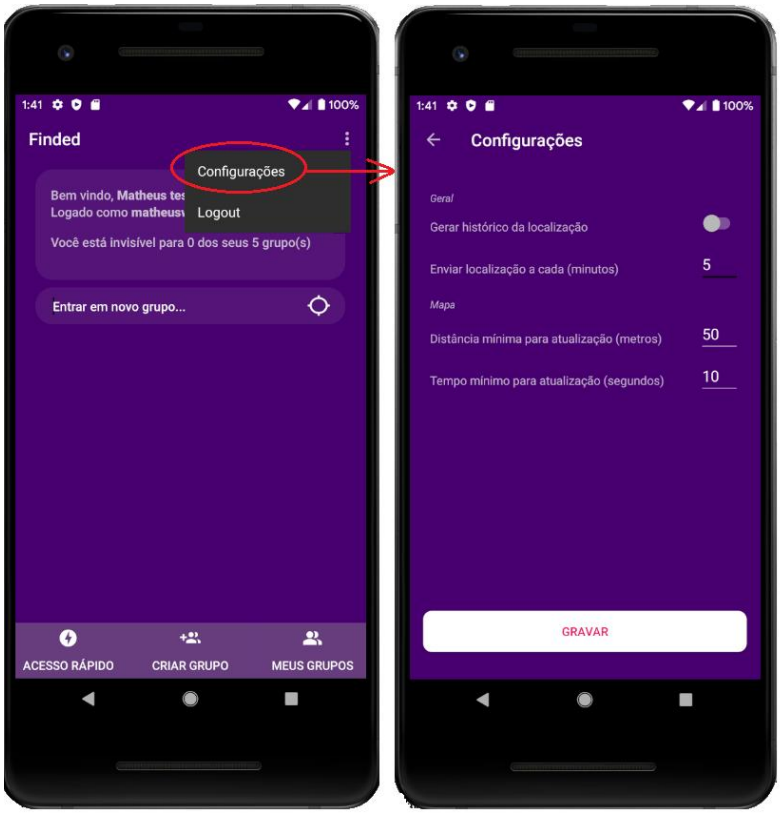
- Enviar localização a cada (*default* 5*)*: define o intervalo de agendamento do AlarmManager, responsável por enviar a localização do usuário em segundo plano. Quanto menor o intervalo, maior o consumo de dados;

- Distância mínima para atualização (*default* 30*)*: define a distância mínima para acionar o *callback* do método *requestLocationUpdates* na Activity Maps (seção 5.5.5);

- Tempo mínimo para atualização (*default* 20*)*: define o tempo mínimo para acionar o *callback* do método *requestLocationUpdates* na Activity Maps (seção 5.5.5).

As configurações são a nível de usuário e ficam armazenadas no Cloud Firestore. Sempre que o aplicativo é iniciado, elas são lidas e na sequência as rotinas de inicialização são executadas com os respectivos parâmetros. A motivação para essa implementação foi para auxiliar nos testes iniciais e oferecer um nível de personalização ao usuário, além de que no futuro alguns destes recursos podem ser limitados em versões *free*. Através da Figura 46 é possível visualizar como acessar a tela de configurações e definir os parâmetros:

Figura 46 – Configurações de execução do aplicativo



Fonte: Do autor (2020).

5.5 Testes

Considerando a diversidade de dispositivos móveis existentes no mercado e as diferentes versões do Android, durante o desenvolvimento o aplicativo foi testado em diferentes aparelhos, visando observar se o comportamento ocorreria conforme o esperado em cada um deles.

Alguns destes dispositivos são físicos e outros são máquinas virtuais (VM) disponibilizadas pelo Android Studio para auxiliar no desenvolvimento. As VMs foram ideais para testar o comportamento das tecnologias em diferentes versões do Android e configurações, tendo a facilidade de executar várias instâncias ao mesmo tempo. Já os aparelhos físicos são essenciais para testar as técnicas de localização e mobilidade, características essas que não são possíveis de testar nas VMs. A Tabela 5 exibe detalhes sobre cada dispositivo testado.

Tabela 5 – Dispositivos utilizados nos testes

Modelo	**Android**	**Localização**	**Tipo**	**Físico/VM**
Motorola Moto G4 XT1621	7.0	A-GPS, GLONASS, BeiDou	Smartphone	Físico
Motorola Moto G6 Play	8.0	A-GPS, GLONASS, BeiDou	Smartphone	Físico
Samsung A SM-T285	5.4	GPS, GLONASS	Tablet	Físico
Google Pixel	5.0	GPS	Tablet	VM
Google Pixel	10	GPS	Tablet	VM
Asus ZenFone 5	9.0	A-GPS, GLONASS, BeiDou	Smartphone	Físico
Samsung Galaxy S10	10	A-GPS, GLONASS	Smartphone	Físico

Fonte: Do autor (2020).

Esta etapa foi importante pois nela foram realizados os primeiros testes com dispositivos físicos que contavam com sensores de localização, visto que até então a maioria dos experimentos era feito com as máquinas virtuais que apenas retornam coordenadas fictícias. Uma das principais melhorias resultantes foi a utilização da classe Criteria ao invés de especificar o provedor de localização desejado.

Durante esta etapa também identificou-se a necessidade de requisitar ao usuário a desativação de otimização de bateria em dispositivos com a versão Android 7 ou superior. Do contrário, os alarmes programados pelo AlarmManager não eram executados pelos motivos descritos na seção 5.4.3.

Após realizar melhorias pontuais e correções de *bugs,* a primeira versão do Finded foi gerada e iniciou-se a etapa de validação do experimento conforme descreve o capítulo 6.

6 VALIDAÇÃO

Após o desenvolvimento do aplicativo foram realizados testes para validá-lo, avaliar sua qualidade e apontar possíveis melhorias. Estes testes foram divididos em duas etapas, sendo a primeira delas relacionado a parte técnica e de *layout*, denominado Teste piloto, e a segunda parte com usuários finais, denominado Testes finais.

Na primeira etapa participaram dois usuários com conhecimento na área de desenvolvimento *mobile* e um com conhecimento na área de *design,* ambos estudantes da instituição. Objetivou-se testar as principais tecnologias do aplicativo em diferentes condições de uso, bem como sua usabilidade e intuitividade.

A segunda etapa consistiu em remeter o aplicativo para validação com os usuários finais, onde dezoito voluntários participaram, entre eles familiares, amigos, colegas da instituição e colegas de trabalho. Por meio destes testes objetivou-se constatar se a proposta do presente trabalho foi atendida conforme as expectativas.

6.1.1 Testes piloto

Realizado por dois especialistas na área de desenvolvimento *mobile*, o teste piloto consistiu em utilizar o aplicativo em diferentes condições e observar o funcionamento das suas principais tecnologias, como autenticação, sincronização de dados com a nuvem e obtenção da localização do dispositivo.

Para realização dos testes, a proposta do aplicativo e as tecnologias envolvidas foram apresentadas aos testadores que nos dias seguintes o utilizaram em diferentes condições, como ambientes *indoor* e *outdoor,* dispositivos com conexões 3G, 4G, Wi-Fi e sob influência de outros *softwares,* como antivírus e gerenciadores de bateria.

Como resultado implementou-se melhorias na execução em *background* para reagendar os alarmes caso o dispositivo seja reiniciado e para testar se há conexão com a *internet* antes de realizar a conversão das coordenadas geográficas em endereços humanamente compreensíveis através da classe Geocoder. Do contrário, se o dispositivo fosse reiniciado ou se a conexão com a *internet* fosse interrompida, a execução em *background* falhava. A partir destes testes também foram feitas algumas melhorias pontuais no comportamento do *chat*, na projeção da posição dos usuários no mapa e nas rotinas de inicialização do aplicativo. Os testes técnicos indicaram o correto funcionamento e desempenho do aplicativo Finded.

Outra situação observada é que nos dispositivos móveis da marca Asus os alarmes não eram executados em segundo plano, mesmo com a não otimização de bateria. Após pesquisas e testes identificou-se que a solução é incluir o aplicativo Finded no “Gerenciador de inicialização” da ferramenta Mobile Manager. Este tipo de situação não há como tratar porque é uma característica de *software* instalado no dispositivo e poderia acontecer também para outros modelos.

Também foram realizados testes de *design* por um especialista na área, buscando avaliar características do aplicativo em relação aos conceitos de Interação Humano Computador (IHC). Para isso as funcionalidades foram apresentadas ao testador, como autenticação, criação de grupos, entrar ou sair de grupos existentes, visualização dos grupos, interação com o mapa, recurso de invisibilidade e troca de mensagens no *chat.*

Como resultado algumas modificações foram feitas no *layout*, tal como a posição dos componentes Progressbar que foram posicionadas mais perto dos botões de ação, o estilo da fonte dos componentes *TextView* que foram suavizados, a posição dos botões da tela principal que tiveram a cor alterada e contornos

removidos para causar sensação de simplicidade e a substituição de *labels* por *hints*.

Na opção de *chat* também foram feitas algumas melhorias, onde aumentou-se as curvas dos *Textviews* que formam os balões das mensagens, a redução das curvas do componente *EditText* que permite a escrita de mensagens para envio e a quebra de linhas para mensagens grandes.

Mesmo que sutis, todas essas mudanças resultaram em uma aparência muito mais confortável e intuitiva ao usuário. Em seguida uma nova versão do Finded foi gerada e iniciou-se a etapa de validação com os usuários finais, a qual será apresentada a subseção a seguir.

6.1.2 Validação com usuários finais

Após implementar as correções e melhorias apontadas no teste piloto, o aplicativo foi remetido para validação com 18 usuários finais que possuem faixa etária entre 18 e 65 anos, alguns com familiaridade no uso de dispositivos móveis e outros não.

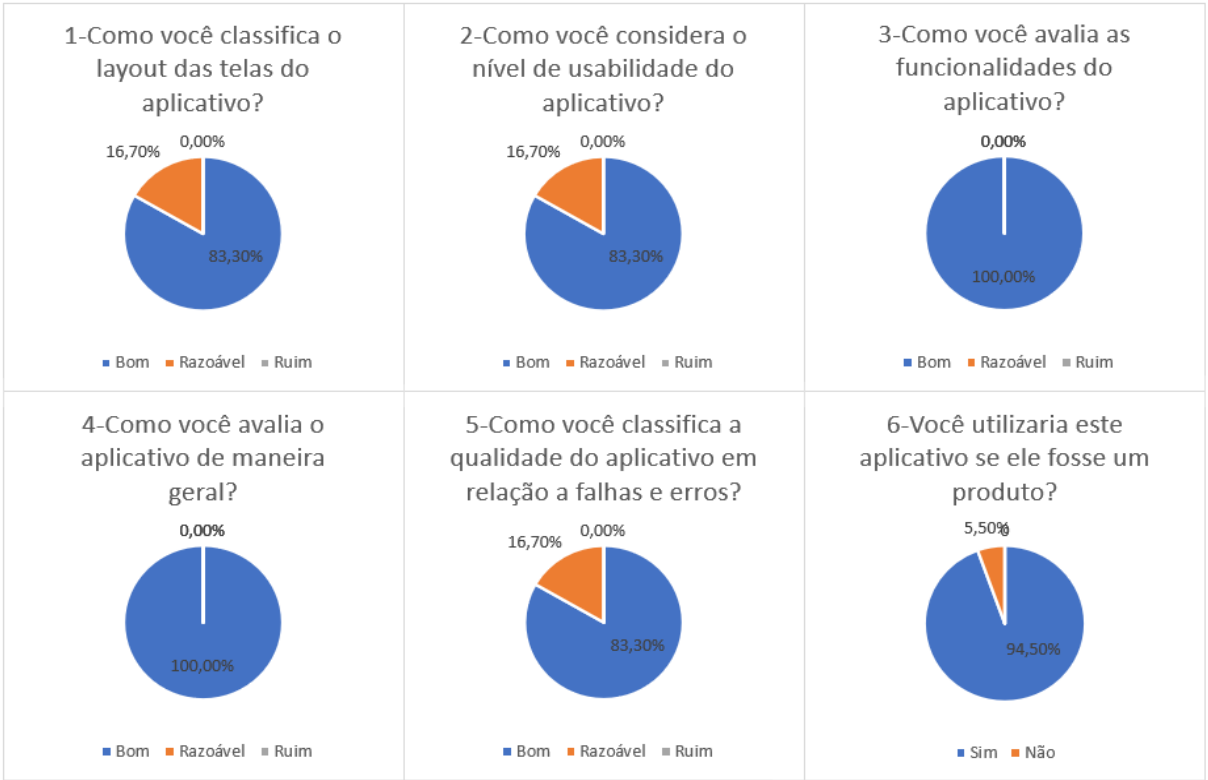
A ideia do aplicativo Finded foi apresentado aos voluntários e a validação foi realizada através de um roteiro para testar as funcionalidades desenvolvidas. O roteiro consistiu nas seguintes tarefas: autenticar-se, criar um grupo e compartilhá-lo por Whatsapp, Facebook ou *e-mail*. Em seguida, quatros usuários entraram no grupo da Família Frey, outros dez no grupo Colegas Cantustange e outros quatro no grupo Colegas Univates. As chaves de acesso e senha de cada grupo foram devidamente enviadas por Whatsapp. Na sequência cada usuário deve consultar os grupos que faz e em seguida entrar no mapa do grupo para visualizar a localização dos membros. Cada usuário também deve interromper o envio da posição em determinado momento do dia, bem como enviar algumas mensagens no *chat*.

Para realização destes testes, nenhum tipo de suporte foi prestado aos voluntários, sendo apresentado apenas as tarefas do teste, justamente para que os usuários pudessem realizá-las intuitivamente.

O teste estendeu-se por dois dias, entretanto alguns voluntários tiveram uma experiência mais profunda. Dois deles instalaram o aplicativo no *smartphone* de seus filhos pré-adolescentes, outro instalou no dispositivo de um familiar que necessita cuidados especiais e um outro voluntário utilizou o aplicativo com um pequeno grupo de amigos que realizou uma viagem intermunicipal.

Ao final dos testes, cada usuário respondeu um questionário para avaliar a qualidade do aplicativo criado. O questionário é baseado na norma ISO 9126 para qualidade de *software* e composto por cinco perguntas a serem respondidas com as opções Bom, Razoável ou Ruim. Uma opção dissertativa também é destinada a críticas e sugestões, além de outra questão para identificar se os usuários utilizariam o aplicativo caso ele fosse um produto do mercado. O resultado deste questionário pode ser observado na Figura 47.

Figura 47 – Resultado da avaliação



Fonte: Do autor (2020).

Alguns usuários mostraram dificuldades para consultar em quais grupos estavam invisíveis. Esta dificuldade estava ligada ao fato de que era necessário entrar nos detalhes do grupo para enxergar essa informação. Para melhorar a

percepção, implementou-se uma sinalização no *Recycler View* responsável por listar os grupos que indica se está visível ou invisível.

Outros usuários com menos familiaridade no uso dos dispositivos móveis mostraram dificuldades na criação de contas para *login* e na autenticação por conta Google. Os mesmos usuários também mostraram dificuldades para realizar *logout.* Apesar das dificuldades, todos usuários conseguiram realizar suas tarefas conforme havia sido proposto.

O resultado da validação foi satisfatório, dado que o aplicativo alcançou os objetivos propostos e nenhum quesito foi classificado como “ruim”. A grande maioria dos *feedbacks* foram positivos e alguns usuários deixaram algumas sugestões que, conforme palavras deles, segue abaixo:

- “Os grupos poderiam ser listados na tela inicial para ser mais rápido acessá-los”;
- “Seria melhor se tivesse a foto de cada membro no mapa ao invés do ícone roxo”;
- “Falta uma opção para remover membros do grupo ou pelo menos para não aceitá-los quando tentarem entrar no grupo”;
- “Poderia ter uma opção para pesquisar lugares no mapa e ver informações sobre o trânsito”;
- “Seria melhor se os outros membros pudessem ver quando eu me escondo. Do contrário, quando estou escondida, parece que não estou no grupo”.

A implementação da primeira sugestão é de baixa complexidade pois já existe uma função que lista os grupos, sendo necessário apenas alterar o *layout* da ActivityMain para utilizá-la.

A segunda sugestão é de média complexidade pois seria necessário criar funções para manipular este tipo de dado. Um facilitador é que o Firebase permite acessar a foto de perfil das contas federadas, como Google, Facebook e Twitter.

A terceira sugestão é de alta complexidade porque seria necessário criar um conceito de administrador e alterar as rotinas que regem os grupos do aplicativo. Consequentemente demandaria bastante tempo para implementar e testar todas as rotinas que garantem a segurança do aplicativo.

O quarto tópico para pesquisar lugares é de complexidade baixa e não foi implementada porque não estava no escopo do projeto. Como alternativa, é possível exportar o Marker do Finded para o Google Maps e a partir disto utilizar recursos como *places*, *traffic* etc.

O último tópico é uma característica do aplicativo Finded, projetado para situações em que apenas alguns membros devem ficar visíveis, como prestadores de serviços e pacientes monitorados. Caso venha a ser um problema para alguns usuários, seria possível implementar uma configuração por grupo para definir este comportamento.

Notou-se portanto que a etapa de validação foi proveitosa e o aplicativo atendeu às expectativas. Alguns dos voluntários estão utilizando o aplicativo para situações reais, como para monitoramento de filhos pré-adolescentes e familiar que necessita cuidados especiais. Constatou-se que o teste piloto também foi importante pois colaborou com a experiência dos usuários finais, resultando em uma avaliação positiva. Os *feedbacks* foram importantes para identificar melhorias que os usuários esperam futuramente e as características que eles mais aprovaram.

7 MELHORIAS FUTURAS

A proposta do aplicativo Finded foi concluída com sucesso, porém algumas melhorias poderiam ser feitas para agregar novos recursos e consequentemente ter um alcance maior no mercado. Seriam elas:

a) Implementar uma solução web para acessar o mapa do grupo e o histórico de localizações do usuário, dado que o Firebase permite compartilhar toda a base de dados entre projetos Web e *Mobile*;

b) Criar uma API REST para receber atualizações sobre a posição de outros membros, sem depender de *smartphones* ou *tablets*. Isso permitiria instalar rastreadores (baseados em ESP32, por exemplo) em veículos ou objetos e integrar com o Finded;

c) Criar configurações para pré-estabelecer o dia e horário que o dispositivo mantém o rastreamento, voltando-se para utilização de empresas que desejam rastrear colaboradores ou prestadores de serviços;

d) Melhorar o sistema de notificações do *chat* para possibilitar responder mensagens pela tela de notificações, sem precisar entrar no aplicativo. Da mesma forma, criar uma Intent para permitir entrar em grupos diretamente pelo convite enviado pelo Whatsapp, Gmail etc.;

e) Realizar melhorias no mapa para possibilitar a pesquisa de endereços e adição dinâmica de Markers, como restaurantes, hotéis, entre outros.

8 CONCLUSÃO

Através da pesquisa realizada identificou-se que as técnicas de localização presentes nos dispositivos móveis são altamente avançadas, tanto em ambientes *indoor* quanto *outdoor*. Da mesma forma, as tecnologias atuais permitem a criação de ferramentas eficientes para auxiliar os usuários em tarefas relacionadas a mobilidade, proporcionando agilidade e eficiência em seu cotidiano.

Aliando estas características com a popularidade dos *smartphones* e *tablets*, foi criado um aplicativo para plataforma Android que explora as técnicas de localização e permite conectar os usuários através de grupos, possibilitando que estes compartilhem suas localizações geográficas em tempo real e troquem mensagens de texto através de um *chat online*.

Levando em conta a infinidade de aplicativos de rastreamento existentes no mercado e inúmeros trabalhos relacionados ao tema, buscou-se desenvolver um aplicativo novo, com características e funcionalidades exclusivas. Por este motivo a dinâmica de grupos foi implementada, permitindo que os usuários acessem mapas privados, apenas com membros de seu interesse. Recursos extras de segurança também foram disponibilizados para usuários autenticarem-se e interromperem o envio da localização quando acharem prudente.

Em relação a implementação do aplicativo, conclui-se que os métodos da plataforma Android para acesso à localização geográfica permitem lidar facilmente com os diferentes ambientes e condições que o dispositivo pode estar submetido, assegurando que a localização seja obtida em praticamente todo território global de

maneira rápida e eficaz. Além disso, estes métodos também possibilitam otimizar o uso de recursos do dispositivo pois contam com funções apropriadas para diferentes situações, como atualizações periódicas, únicas, aproximadas, precisas, detecção de movimento, entre outras. Desta maneira, é possível implementar as rotinas do aplicativo para usar as funções mais adequadas levando em conta a necessidade de cada funcionalidade desenvolvida.

Já o Firebase mostra-se ideal para projetos de curto prazo, como é o caso das *startups*. Essa característica está relacionada com a diversidade de recursos oferecidos e porque a ferramenta absorve toda parte do *backend*, oferecendo serviços eficientes e seguros aos desenvolvedores que não precisam investir tempo para implementá-los e nem recursos financeiros para construir a infraestrutura. Na implementação do aplicativo Finded, por exemplo, utilizou-se o Firebase para gerenciar suas principais funcionalidades, como autenticação, persistência de dados, sincronização com a nuvem e atualizações em tempo real, bem como geração e exibição de notificações *push*.

Entre os recursos utilizados, destaca-se o Cloud Firestore, que provou ser adequado para este tipo de aplicação que requer atualizações em tempo real, como é o caso das coordenadas e mensagens do *chat*. Sua arquitetura projetada para realizar tarefas em *background* e acionar um *callback* após a conclusão causa a sensação de leveza por não sobrecarregar *UI thread*, sendo este um aspecto relevante no desempenho e usabilidade.

Os resultados obtidos neste trabalho foram satisfatórios, tanto em relação ao funcionamento quanto ao desempenho do aplicativo Finded. Para comprovar esta afirmação pode ser considerado o resultado dos testes de validação, onde há um alto índice de aprovação e a grande maioria dos usuários afirmou que usaria o aplicativo caso ele fosse disponibilizado como um produto.

Por fim, constata-se que o aplicativo Finded é capaz de aproximar os usuários e está apto para auxiliar aqueles que necessitam ter acesso sobre informações de localização de outras pessoas. Estima-se que com a implementação das melhorias futuras, cresce a oportunidade de entrar no mercado como um produto.

REFERÊNCIAS BIBLIOGRÁFICAS

AGÊNCIA BRASIL. **Mais de 5 bilhões de pessoas usam aparelho celular, revela pesquisa.** 2019. Disponível em: <https://agenciabrasil.ebc.com.br/geral/noticia/2019-09/mais-de-5-bilhoes-de-pessoas-usam-aparelho-celular-revela-pesquisa>. Acesso em: 21 abr 2020.

ALVES, Sérgio. **A matemática do GPS**. Revista do Professor de Matemática, v. 59, p. 17-26, 2006. Disponível em <ftp://labattmot.ele.ita.br/ele/jacques/CursoNaveg/gps.pdf>. Acesso em: 25 março 2020.

ANDROID DOCUMENTATION. Disponível em <https://developer.android.com/guide/platform?hl=pt>. Acesso em: 03 março 2020.

ARON, Lukas; HANÁCEK, Petr. Introduction to Android 5 Security. In: SOFSEM (Student Research Forum Papers/Posters). 2015. p. 103-111.

BOHRER, Fernando José. **Serviço de geolocalização para plataforma Android**. 2011. Trabalho de conclusão de curso. Centro Universitário Univates. Lajeado. RS.

BRAGA, Felipe Augusto; DA SILVA, Marcos Alberto Lopes. **Implementação de Serviços em Nuvem Baseado no Conceito de Serverless**. e-RAC, v. 8, n. 1, 2018.

CAMILO, Gabriel Magalhães Menezes. **Sistema web para controle de vacinação de um hospita**l. 2019. 47 f. Monografia (Graduação em Sistemas de Informação) - Instituto de Ciências Exatas e Aplicadas, Universidade Federal de Ouro Preto, João Monlevade - MG, 2019.

CASSONI, Luis Aurélio; SGARBI, Nara Maria Fiel de Quevedo; SAKAGUTI, Solange Tieko. Smartphones e suas utilidades usuais e educacionais. **INTERLETRAS.** Inns nº 1807-1597. V. 6, Edição número 24, Outubro de 2016 a Abril 2017.

CERQUEIRA. **Mundo educação.** Disponível em: <http://mundoeducacao.bol.uol.com.br/>. Acesso em: 14 mai 2016.

CHIZZOTTI, Antonio. **Pesquisa em ciências humanas e sociais**. Cortez editora, 2018. Disponível em. Ebook.

COUTINHO, Gustavo Leuzinger. **A era dos *smartphones:* Um estudo exploratório sobre o uso de *smartphones* no Brasil.** 2014. Monografia. Universidade de Brasília. Brasília – DF, 2014.

DANIELI, Rafael Leonardo. **Sistema gerador de rotas através de mapas de calo**r. 2015. Monografia (Graduação em Engenharia da Computação) – Universidade do Vale do Taquari - Univates, Lajeado, 27 nov. 2015. Disponível em: <http://hdl.handle.net/10737/921>.

DATE, Christopher J. **Introdução a sistemas de bancos de dados**. E-Book. Elsevier Brasil, 2004.Tradução da 8ª Edição Americana.

DILIÃO, Rui. **GPS: Global Position System.** Disponível em: <http://www.cienciaviva.com/latlong/anterior/gps.asp>. Acesso: 25 março 2020.

DE ARAÚJO, Gregory Eilert. **POLITER – Aplicativo android voltado a interesses políticos**. Projeto Aplicado para Projetista-Projeto Integrador, v. 1, n. 1, 2018.

DO LAGO, Isabel Franco; FERREIRA, Luiz Danilo Damasceno; KRUEGER, Claudia Pereira. **GPS e GLONASS: aspectos teóricos e aplicações práticas**. Boletim de Ciências Geodésicas, v. 8, n. 2, 2002.

DOCUMENTATION GOOGLE MAPS API. **Google Maps Plataform.** Disponível em: <https://cloud.google.com/maps-platform>. Acesso em: 12 mai 2020.

DOCUMENTATION ANDROID TASK. **Task.** Disponível em <https://developer.android.com/guide/components/activities/tasks-and-back-stack>. Acesso em: 12 mai 2020.

DOCUMENTATION ANDROID ALARMMANAGER. **AlarmManager.** Disponível em: <https://developer.android.com/reference/android/app/AlarmManager>. Acesso em: 12 mai 2020.

DOCUMENTATION ANDROID SNAPSHOT LISTENER. **Receber atualizações em tempo real com o Cloud Firestore.** Disponível em: <https://firebase.google.com/docs/firestore/query-data/listen?hl=pt-br>. Acesso em: 25 mai 2020.

DOCUMENTATION ANDROID BACKGROUND. **Create a background service.** Disponível em <https://developer.android.com/training/run-background-service/create-service>. Acesso em: 12 mai 2020.

DOCUMENTATION FIREBASE AUTH. **Firebase Authentication.** Disponível em <https://firebase.google.com/docs/auth>. Acesso em: 12 mai 2020.

DOCUMENTATION FIREBASE CLOUD FUNCTIONS. **Cloud Functions para Firebase.** Disponível em <https://firebase.google.com/docs/functions?hl=pt-PT>. Acesso em 26 out 2020.

DOCUMENTATION FIREBASE CLOUD MESSAGING. **Sobre as mensagens do FCM.** Disponível em < https://firebase.google.com/docs/cloud-messaging/concept-options?hl=pt-br>. Acesso em 26 out 2020.

DOCUMENTATION FIREBASE FIRESTORE. **Cloud Firestore.** Disponível em <https://firebase.google.com/docs/firestore?hl=pt-br>. Acesso em: 12 mai 2020.

FIGUEIREDO, Davis Anderson. **Análise de vulnerabilidades e ameaças presentes em redes Wi-Fi (ieee 802.11) de instituições de ensino superior de Minas Gerais**. 2016. Projetos e Dissertações em Sistemas de Informação e Gestão do Conhecimento, v. 5, n. 2, 2017.

GARTNER, Gartner. **Gartner Says Worldwide Smartphone Sales Will Grow 3% in 2020.** 2020. Disponível em: < https://www.gartner.com/en/newsroom/press-releases/2020-01-28-gartner-says-worldwide-smartphone-sales-will-grow-3-->. Acesso em: 21 abr 2020.

GIL, Antonio C. **Como elaborar projetos de pesquisa.** 4. ed. São Paulo: Atlas, 2002.

IDC, International Data Corporation. **Smartphone Challenges Continue in 2019, But 5G and Emerging Markets Will Bring Growth Back to the Market in 2020, According to IDC.** 2019. Disponível em: <https://www.idc.com/getdoc.jsp?containerId=prUS45487719>. Acesso em: 21 abr 2020.

JÚNIOR, Paulo de Tarso Setti; ALVES, Daniele Barroca Marra; GOUVEIA, Tayná Aparecida Ferreira. Uso integrado dos sistemas Galileo e GPS: uma análise da acurácia no posicionamento por ponto com correções atmosféricas. **Revista Brasileira de Cartografia**, v. 68, n. 3, 2016

LIMA, Erly Caldas de. **Proposta de metodologia para melhora do posicionamento obtido através de receptores GPS de baixo custo**. 2018. Tese de Doutorado. Universidade de São Paulo – São Paulo, SP.

LECHETA, Ricardo R. **Google Android-3ª Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. Novatec Editora, 2013.

LUMMERTZ, Ramon Santos; SGANZERLA, Antoni. **Direto ao Ponto–App colaborativo do transporte coletivo usando o Firebase**. Conversas Interdisciplinares, v. 14, n. 1, 2018.

MACHADO, Everto Fábio da Silva. **Desenvolvimento de sistemas de geolocalização e rastreamento para plataforma Android – COMPASS.** Trabalho de conclusão. Universidade Tecnóloga Federal do Paraná, Francisco Beltrão - PR, 2015.

LAKATOS, Eva M.; MARCONI, Marina de A. **Ciência e conhecimento científico. Fundamentos da metodologia cientifica**. São Paulo: Atlas, p. 74-81, 2001.

MATIAS, Diogo Francelino. **Protótipo de aplicativo móvel para localização de estabelecimentos de alimentação sem ponto fixo**. Aplicativo Móvel, 2019. Disponível em <http://repositorio.satc.edu.br/bitstream/satc/371/2/TCC%20Diogo%20pdf.pdf>. Acesso em: 13 abr 2020.

MENEZES, Raiane Rintielle Vaz. **Análise do GNSS PPP multi-constelações com uso dos sistemas GPS, GLONASS E GALILEO**. 2019. Tese de Doutorado. Universidade Federal de Viçosa - MG.

MONICO, João F. G. **Posicionamento pelo NAVSTAR-GPS:** Descrição, fundamentos e aplicações. São Paulo: Unesp, 2000.

MOTA, Vitor Luiz Gomes; CARVALHO, Roberta; CORREA, Carina; RENNA, Roberto Brauer Di; MAGRI, Vanessa; FERREIRA, Tadeu; CASTELLANOS, Pedro; MATOS, Leni. **Evolução da tecnologia de telefonia móvel e estudo e caracterização de um sistema móvel 5G de quinta geração.** Artigo UFF. 2019.

MOURA, André Iasi. **WBLS: um sistema de localização de dispositivos móveis em redes Wi-Fi**. 2007. Tese de Doutorado. Universidade de São Paulo.

NASA, National Aeronautics and Space Administration. **What is GPS?** Disponível em <https://www.nasa.gov/sites/default/files/atoms/files/gps_web.pdf>. Acesso em: 25 março 2020.

NERIS JR, Celso; FUCIDJI, José Ricardo; GOMES, Rogério. Trajetórias tecnológicas da indústria de telefonia móvel: um exame prospectivo de tecnologias emergentes. **Economia e Sociedade**, v. 23, n. 2, p. 395-431, 2014.

PEREIRA, Lucio Camilo Oliva; DA SILVA, Michel Lourenço. **Android para desenvolvedores**. Brasport, 2009.

POLON, Luana. **Coordenadas geográficas**. Disponível em: <https://www.estudopratico.com.br/coordenadas-geograficas/>. Acesso em: 25 março 2020.

PONTES, Lucas Andrade. **Mobilenha: aplicativo para cálculo estimativo de consumo de lenha e da produção na indústria de cerâmica vermelha**. 2019. Trabalho de Conclusão de Curso. Universidade Federal do Rio Grande do Norte.

PRODANOV, Cleber Cristiano; DE FREITAS, Ernani Cesar. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico** - 2ª Edição. Editora Feevale, 2013.

RICARDO, Solange Cristina; SILVEIRA, Árcio. **A evolução do sistema da tecnologia de telefonia móvel como geradora de comunicação ubíqua e pervasiva.** Anais do V SINGEP – São Paulo – SP – Brasil – 20, 21 e 22/11/2016 . Disponível em < https://singep.org.br/5singep/resultado/476.pdf>. Acesso em: 30 mar 2020.

RAMOS, Daniel Souza. Uso de um banco de dados orientado a documentos para suporte a aplicações de big data. **Sistemas de Informação-Pedra Branca**, 2014.

SALADWGE, Pramod J.; FOWLER, Martin. **NOSQL Essencial: Um Guia Conciso para o Mundo Emergente de Persistência Poliglota**. São Paulo: Novatec, 2013.

SANTOS, Marco Aurélio dos et al. **Localização em ambientes internos utilizando PDR e Wi-Fi**. 2018. Dissertação. Universidade Federal do Amazonas. Manaus – AM. 2018.

SCHONARTH, Murilo. **FINDAY: Aplicativo para auxiliar a mobilidade em ambientes *indoor*.** 2017. Dissertação. Universidade do Vale do Taquari – Univates, RS. 2017.

SILVA, Ítala Liz Da Conceição Santana; **Do 1G Aa 5G: Evolução das redes de telefonia móvel.** 2016. Monografia. Universidade Federal do Recôncavo da Bahia. Cruz das Almas, BA. 2016.

SILVA, Paulo et al. **Estudo comparativo entre bancos de dados SQL e NoSQL**. 2019. Plataforma de Submissão de Trabalhos e Anais de Eventos da Unicruz, 2019.

SILVA, Caio Poli; FEYH, Pedro Gustavo Ramm; ROLAND, Carlos Eduardo de França. **PETS: desenvolvimento de sistema de geolocalização para monitoramento de animais de estimação. 2018.** Revista eletrônica de sistemas de informação e gestão tecnológica, Uni-FACEF, Vol. 9, Nro 3, 2018.

SILVEIRA, Denise Tolfo; CÓRDOVA, Fernanda Peixoto. Unidade 2 – A pesquisa científica. **Métodos de pesquisa**, v. 1, 2009. Disponível em < https://www.cesadufs.com.br/ORBI/public/uploadCatalago/11315818082016Pratica_de_Pesquisa_I_Aula_2.pdf>. Acesso em: 13 abr 2020.

SOMMERVILLE, Ian. **Engenharia de software.** 8. ed. São Paulo: Pearson Education do Brasil Ltda, 2007. E-book. Disponível em: <https://www.univates.br/biblioteca/>. Acesso em 16 abr. 2020.

SOUSA, Kássio Rômulo Lima. **Sistema colaborativo para criação de roteiros turísticos e Interativos utilizando a API Google Maps**. TCC Graduação em Ciência da Computação. Universidade Federal do Maranhão. São Luís – MA. 2016.

TERRERI, Rafael Grimming; DOS SANTOS, Euripedes Ramos; DA SILVA, Marcos Alberto Lopes. **Proposta de solução para implementação de um sistema de identificação e notificação de presença de alunos usando as plataformas: android, arduino e firebase.** e-RAC, v. 8, n. 1, 2018.

VAZ, Jhonnes Alberto; PISSARDINI, Rodrigo de Souza; JUNIOR, Edvaldo Simões da Fonseca. **Comparação da cobertura e acurácia entre os sistemas GLONASS e GPS obtidas dos dados de observação de uma estação da rede brasileira de monitoramento contínuo**. Revista Brasileira de Cartografia, n. 65/3, 2013.

ZANDBERGEN, Paul A; BARBEAU, Sean J. **Positional Accuracy of Assisted GPS Data from High-Sensitivity GPS-enabled Mobile Phones.** THE JOURNAL OF NAVIGATION (2011), 64, 381–399. 2011. Disponível em <https://www.researchgate.net/publication/231849997_Positional_Accuracy_of_Assisted_GPS_Data_from_High-Sensitivity_GPS-enabled_Mobile_Phones>. Acesso em: 26 mar 2020.

ZENLY, Oficial Web Site. Disponível em <https://zen.ly/>. 2020.