PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

GEOFENCING APLICADO À COMERCIALIZAÇÃO DE PRODUTOS ELETRÔNICOS EM PLATAFORMA *MOBILE*

PEDRO GUILHERME PEREIRA DOS REIS

GOIÂNIA
2021

PEDRO GUILHERME PEREIRA DOS REIS

GEOFENCING APLICADO À COMERCIALIZAÇÃO DE PRODUTOS ELETRÔNICOS EM PLATAFORMA *MOBILE*

Trabalho de Conclusão de Curso apresentado à Escola Politécnica, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador:
Prof. M.E.E. Marcelo Antonio Adad de Araújo

GOIÂNIA

2021

PEDRO GUILHERME PEREIRA DOS REIS

GEOFENCING APLICADO À COMERCIALIZAÇÃO DE PRODUTOS ELETRÔNICOS EM PLATAFORMA *MOBILE*

Trabalho de Conclusão de Curso aprovado em sua forma final pela Escola Politécnica, da Pontifícia Universidade Católica de Goiás, para obtenção do título de Bacharel em Engenharia de Computação, em 08 de dezembro de 2021.

_____

Prof. MSc. Ludmilla Reis Pinheiro dos Santos
Coordenadora de Trabalho de Conclusão de Curso

Banca examinadora:

_____

Orientador: Prof. M.E.E. Marcelo Antonio Adad de Araújo

_____

Prof. MSc. Pedro Araújo Valle

_____

Prof. M.E.E. Carlos Alexandre Ferreira de Lima

GOIÂNIA
2021

A Deus, por ter me capacitado no desenvolvimento deste trabalho.

Aos meus pais, pela educação, amor e cuidado ao longo de toda a minha vida.

Ao meu tio e à minha vó, sem eles este trabalho não seria possível.

Aos meus familiares que estiveram sempre presentes em minha vida.

AGRADECIMENTOS

A Deus por me proporcionar a benção de poder realizar um curso superior, e me manter perseverante na elaboração deste trabalho, me trazendo foco e me direcionando no caminho correto.

Aos meus pais Gunther Guilherme do Prado Reis e Volga Pereira de Santana Reis pelo apoio e amor incondicional ao longo da minha vida, me proporcionando uma boa educação tanto dentro quanto fora de casa, e me auxiliando, orientando e instruindo da melhor forma possível.

Ao meu tio Dulcimar Pessatto Filho e à minha vó Nadya Santana Pereira por me amarem e acreditarem em mim, investindo na minha educação dentro desta instituição desde o início, me possibilitando fazer um curso superior, e futuramente honrá-los sendo um bom profissional.

Ao meu professor orientador Marcelo Antonio Adad de Araújo pelo auxílio, paciência e confiança ao longo de todo o ano, me instruindo de forma a realizar este trabalho com excelência.

Ao meu amigo Douglas Vieira do Nascimento, por confiar em meu trabalho, me proporcionando um excelente estágio trabalhando juntos, pela parceria em diversas matérias da faculdade, e por me ajudar, contribuindo para o meu crescimento acadêmico.

Ao meu amigo Shirlano Cândido Dias Filho, pela amizade e companheirismo ao longo de vários anos, me acompanhando em grande parte de minha trajetória acadêmica.

RESUMO

Este trabalho apresenta o desenvolvimento de um aplicativo móvel para o sistema operacional *Android*, que se utiliza da geolocalização, seja do usuário ou de um local determinado pelo mesmo, inserido em uma cerca virtual denominada *geofence*, para realizar buscas de estabelecimentos comerciais que vendam produtos eletrônicos pesquisados pelo próprio usuário. Para isso, este aplicativo é desenvolvido utilizando o banco de dados do *Google Firebase* e a *Integrated Development Environment* (IDE), ou Ambiente de Desenvolvimento Integrado *Android Studio*.

Palavras-Chave: *Android; geofence; Firebase;* geolocalização.

ABSTRACT

This work presents the development of a mobile application for the Android operating system, which uses geolocation, either from the user or from a location determined by the user, inserted in a virtual fence called geofence, to carry out searches of commercial establishments that sell electronic products searched. by the user himself. For this, this application is developed using the Google Firebase database and the Integrated Development Environment (IDE) Android Studio.

Keywords: *Android; geofence; Firebase; geolocation.*

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

M.E.E.	Mestre em engenharia elétrica
MSc.	*Master of Science*
Prof.	Professor

LISTA DE SIGLAS

GPS	*Global Positioning System*
IDE	*Integrated Development Environment*
API	*Application Programming Interface*
Wi-Fi	*Wireless Fidelity*
SQL	*Structured Query Language*
NoSQL	*Not Only Structured Query Language*
JSON	*JavaScript Object Notation*
FCM	*Firebase Cloud Messaging*
RAD	*Rapid Application Development*
DoD	Departamento de Defesa dos Estados Unidos
SDK	*Software Development Kit*
XML	*eXtensible Markup Language*
APK	*Android Aplication Package*
ART	*Android Runtime*

SUMÁRIO

1 INTRODUÇÃO

Este trabalho de conclusão de curso tem por finalidade o desenvolvimento de um aplicativo móvel para *Android* que seja capaz de realizar buscas por produtos eletrônicos vendidos em locais que estejam dentro de uma cerca virtual conhecida como *geofence*. Esta *geofence* será implementada para realizar buscas se baseando na geolocalização do usuário, ou em uma área determinada pelo mesmo. A busca consistirá em encontrar produtos eletrônicos em lojas que vendam este produto, e poderá ser filtrada de acordo com o valor do produto, distância até o local e características específicas de cada produto, como a capacidade de armazenamento de um disco rígido (HD).

O aplicativo conta com um banco de dados, no qual possui informações sobre os produtos eletrônicos, como imagens, preço, local de venda e características específicas de cada produto. O uso da geolocalização possibilita encontrar qualquer ponto dentro no globo terrestre, permitindo ao aplicativo encontrar qualquer estabelecimento que venda o produto desejado, desde que este esteja dentro do raio da *geofence* implementada. O aplicativo é desenvolvido através da plataforma *Android Studio*, que faz uso da linguagem de programação *Java*. Este aplicativo deverá ser capaz de buscar o produto eletrônico pesquisado pelo usuário dentro de uma *geofence*, e trazer até ele localizações de estabelecimentos que possuam este produto a venda, além de informações retrocitadas sobre o produto buscado.

1.1 Objetivos

Esta seção tem por finalidade apresentar e descrever os objetivos gerais e específicos deste trabalho.

1.1.1 Objetivo Geral

Desenvolver um aplicativo *Android* com base na tecnologia *geofencing* para realizar buscas por produtos eletrônicos, tomando como base a localização do usuário com o intuito de trazer, na tela do seu *smartphone*, as lojas que possuem o produto buscado, mostrando também o preço, a localização, um raio de distanciamento determinado pelo usuário e informações específicas do produto.

1.1.2 Objetivos Específicos

- Aplicar o *Geofencing* a automação comercial.
- Implementar um aplicativo *Android* utilizando *Geofencing*.
- Alimentar um banco de dados a ser gerenciado pelo aplicativo.
- Desenvolver o sistema de acordo com as necessidades de compra do usuário.
- Desenvolver o sistema para que o usuário tenha acesso aos produtos na distância determinada pelo mesmo.

1.2 Justificativa

O presente trabalho justifica-se por possibilitar a integração de *Geofencing* à automação comercial utilizando-se de tecnologias que estão presentes nos atuais *smartphones Android*.

O aplicativo em questão, trará para o usuário uma maior comodidade e simplicidade, possibilitando a ele uma facilidade no momento em que quiser realizar a compra de um determinado produto eletrônico, de forma que não precise desprender de muito de seu tempo ao realizar várias pesquisas de mercado, no que tange à buscar preços e localizações que melhor atendam as expectativas para a comercialização do produto buscado, bastando apenas realizar uma pesquisa no aplicativo georreferenciado em questão, e já terá vários resultados de preço, modelos e lojas onde vendam este produto, baseando-se na localização atual do usuário.

1.3 Metodologia

Inicialmente é realizado uma revisão bibliográfica baseada em artigos científicos, livros, revistas, monografias, e vários outros meios de informação que sejam confiáveis. Enquanto isso, também é desenvolvida a construção de um banco de dados relacional que é implementado na plataforma *Google Firebase*, alimentando-o com imagens de produtos, preços, localizações, e outras informações que forem específicas de cada produto. Após isso, implementar-se-á o aplicativo, o qual é desenvolvido por meio da IDE *Android Studio*.

1.4 Resultados Esperados

Espera-se com este trabalho que os resultados obtidos contribuam para o desenvolvimento, crescimento e aprimoramento das soluções de tecnologia voltadas para a área comercial, especialmente as que envolvem *geofencing*, com ênfase em automação comercial junto a aplicativos móveis.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo tem por finalidade, apresentar a fundamentação teórica necessária para a concepção e elaboração deste trabalho, visando a orientação e compreensão a respeito dos conceitos e definições utilizados no desenvolvimento do aplicativo *Android*, no qual se baseia o objetivo deste trabalho.

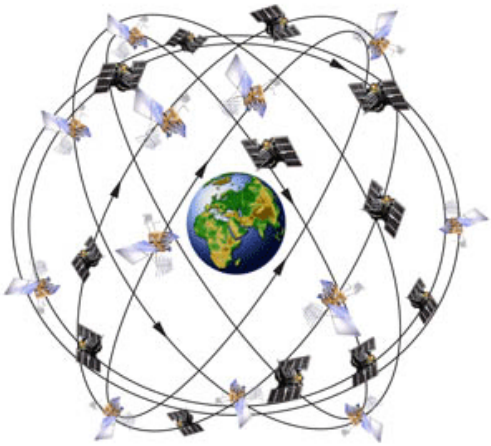
2.1 Geolocalização

Trata-se de um recurso utilizado para se obter a posição no globo terrestre de acordo com o sistema de coordenadas. Para se obter essas coordenadas, esse recurso capta sinais que permitam ter acesso a essa informação, como *Wi-Fi* e o *Global Positioning System* (GPS), ou Sistema de Posicionamento Global (BRANDÃO, 2019).

O GPS é provavelmente o sistema mais popular utilizado para o fim de obtenção de coordenadas no globo terrestre. Ele funciona através de dados recebidos via satélite, os quais se encontram em órbita do planeta Terra. Para poder determinar a localização de um objeto que faça uso dessa tecnologia, ele necessita de receber dados de ao menos 3 satélites, para assim realizar a triangulação de dados e determinar com precisão, a localização geográfica deste objeto (BRANDÃO, 2019).

Este sistema de geolocalização foi desenvolvido pelo Departamento de Defesa dos Estados Unidos (DoD) por volta de 1970, e o primeiro satélite foi lançado em 1978. A constelação de satélites que fornece estes dados de localização, só começou a operar completamente em 1993, esta constelação tem o nome de *NAVSTAR*, e quando foi ativada, possuía 24 satélites. Atualmente, a constelação possue 32 satélites em funcionamento em 6 planos orbitais como demonstrado na Figura 1, com 24 em operação e 8 de reserva, caso haja algum incidente com os satélites operacionais (FAGGIAN, 2019).

Figura 1 – Constelação de satélites ao redor do planeta Terra.



Fonte: Dempsey, 2011.

Todos os 32 satélites são monitorados na Terra por meio de bases fixas, que recebem sinais desses satélites constantemente. Os sinais contêm uma assinatura codificada do satélite que os enviou, o horário exato em que foi emitido e a posição do satélite. Estes satélites são equipados com relógios atômicos que estão sincronizados, e que, é estimado que possuam o erro de um segundo em trezentos mil anos, que acaba se tornando um erro desprezível, já que os referidos relógios são atualizados constantemente (FAGGIAN, 2019).

O GPS determina a posição de onde um receptor se encontra através da triangulação, se baseando na distância entre 3 satélites e o próprio receptor. Assim que é determinada a distância relativa de cada satélite em relação ao receptor, é possível saber em que local do globo terrestre que este receptor está (SANTANA *et.al.*, 2020). Para se determinar o local do receptor, o mesmo se utiliza da hora em que os sinais foram emitidos pelo satélite para calcular a distância, se baseando em quanto tempo levou para sinal de cada satélite chegar até ele (FAGGIAN, 2019).

De acordo com Santana *et. al.* (2020), por conta de sua popularização e grande acessibilidade, o GPS foi implantado também em plataformas *mobile*, como *smartphones*, e é utilizado por meio de aplicativos que fazem uso da geolocalização, como *Google Maps*. O sistema de GPS de *smartphones* pode ser configurado em 3 modos diferentes:

- Alta precisão: faz uso das informações do receptor GPS que está no *smartphone*, somadas às informações de localização obtidas por meio de dados móveis e *Wi-Fi*.
- Economia de Bateria: utiliza apenas os dados móveis e de *Wi-Fi* para determinar a localização.

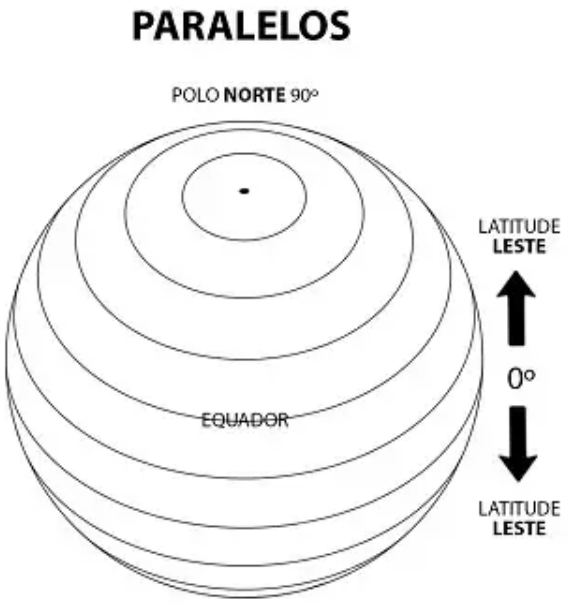
- Somente o aparelho: utiliza apenas as informações fornecidas pelo receptor GPS do *smartphone* para determinar a localização.

Esses modos são ativados de acordo com a preferência do próprio usuário, mas nem sempre um aplicativo fará uso de todos os recursos disponíveis no modo de Alta precisão, e terão outros que só funcionarão da maneira desejada, caso o *smartphone* esteja configurado para utilizar o modo de Alta precisão. Neste caso é o sistema operacional do *smartphone* que irá definir o modo no qual o aplicativo necessita que esteja ativado para que o seu funcionamento ocorra da maneira esperada (SANTANA, *et. al.*, 2020).

Para que o GPS consiga localizar um ponto no globo terrestre, ele utiliza o sistema de coordenadas geográficas. Este sistema faz uso de linhas imaginárias que circundam o planeta Terra, conhecidas como paralelos e meridianos, que por sua vez, possibilitam o uso de códigos geográficos chamados de latitude e longitude (TEMBHEKAR; SAKHARE, 2021).

A latitude consiste na distância angular que é medida a partir da Linha do Equador, indo de norte a sul, sendo que a Linha do Equador sinaliza latitude de 0 graus. Essas linhas imaginárias circundam o globo terrestre de leste à oeste, paralelamente a Linha do Equador, como apresentado na Figura 2. Os graus medidos podem ir de 0 a 90 graus positivos caso a latitude medida se encontre ao norte da Linha do Equador, ou negativos se medida ao sul desta linha (TEMBHEKAR; SAKHARE, 2021).

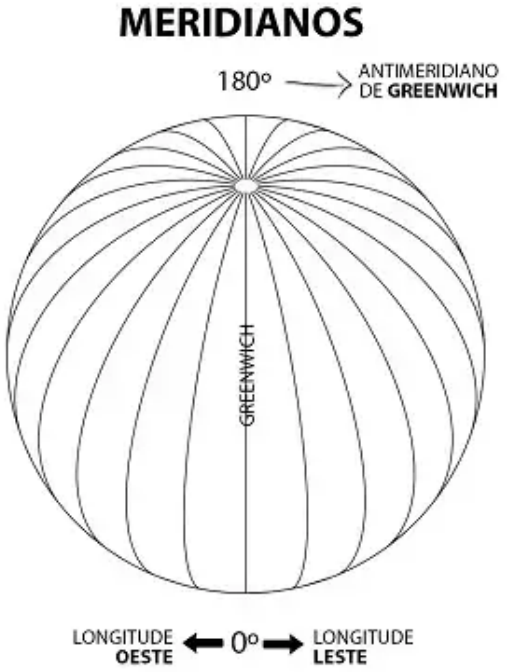
Figura 2 – Linhas latitudinais conhecidas como Paralelos.



Fonte: Adaptado de Polon, 2018.

A longitude, por outro lado, consiste na distância angular medida a partir do primeiro meridiano, ou Meridiano de Greenwich, sendo que o Meridiano de Greenwich possui longitude igual a 0 graus. Neste caso, estas linhas imaginárias percorrem o globo terrestre de forma vertical, indo de norte ao sul, como apresentado na Figura 3. Por conta de o planeta Terra possuir um formato esférico, é considerado que ele possua 360 graus, portanto, o globo terrestre foi divido em 360 meridianos ou linhas longitudinais como forma de medição. Os graus medidos variam entre 0 e 180 graus positivos para longitudes medidas a leste do Meridiano de Greenwich, e negativos para medições a oeste do mesmo (TEMBHEKAR; SAKHARE, 2021).

Figura 3 – Linhas longitudinais conhecidas como Meridianos.



Fonte: Adaptado de Polon, 2018.

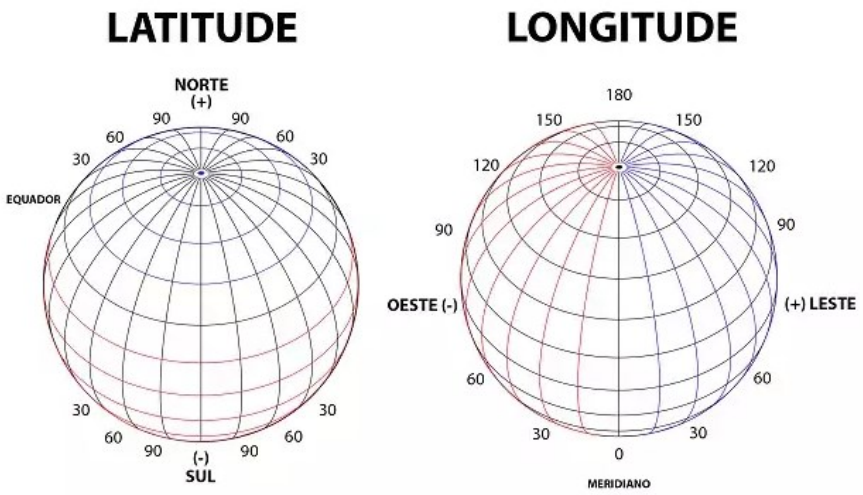
Latitudes e longitudes, quando combinadas, formam um sistema conhecido como sistema de coordenadas geográficas. O GPS se utiliza deste sistema para localizar um ponto específico no globo terrestre. Como o grau, seja em latitude ou longitude, representa uma distância muito grande no globo terrestre, foram realizadas subdivisões do grau para uma medição mais precisa. O grau pode ser medido em decimal, com várias casa após a vírgula. Essa medição é uma das opções usadas no GPS, como também pode ser medido em minutos e segundos, sendo que, 1 grau equivale a 60 minutos, e 1 minuto equivale a 60 segundos (TEMBHEKAR; SAKHARE, 2021).

Como as latitudes e longitudes são dadas em graus, minutos e segundos, estes valores terão de ser convertidos para valores decimais caso precisem ser lidos por sistemas computacionais. Seguindo a lógica de 1 grau possuir 60 minutos, 1 grau possuirá então 3600 segundos, e para realizar a conversão, é utilizada a equação dada por (PRINA; TRENTIN, 2017):

$$Grau_{Decimal} = \left(Graus + \frac{Minutos'}{60} + \frac{Segundos''}{3600}\right) \quad (1)$$

A Figura 4 demonstra como o globo terrestre aparenta após a combinação das linhas imaginárias latitudinais e longitudinais, além de mostrar alguns dos ângulos das latitudes e longitudes.

Figura 4 – Globo terrestre com linhas latitudinais e longitudinais.



Fonte: Adaptado de Polon, 2018.

2.2 *Geofencing*

O *geofencing* consiste em um serviço, implementado em um aplicativo ou *software* computacional, baseado em localização, que se utiliza de GPS, dados de celular e *Wi-Fi*, para realizar uma certa ação previamente programada para quando um *smartphone* adentra ou sai de uma área geograficamente determinada, ou de uma cerca geográfica virtual, que é denominada *geofence* (WHITE, 2017).

A *geofence* pode ser configurada para realizar diversos tipos de ações, como enviar mensagens de texto, notificações *push*, enviar dados de uso de programas dentro da região para diversas empresas, especialmente empresas de marketing (WHITE, 2017).

A *geofence* pode ser usada até mesmo para monitorar determinadas áreas de segurança, emitindo alertas quando um dispositivo entra ou sai desta área, ou até permitindo certas ações específicas dentro de um determinado aplicativo ou programa, quando este é executado em um dispositivo que se encontra dentro ou fora da cerca virtual (WHITE, 2017).

Para fazer uso do *geofencing*, primeiro um desenvolvedor estabelece uma cerca virtual ao redor de uma localização específica, utilizando um programa que faça uso de GPS. A cerca virtual, ou *geofence*, irá então realizar alguma ação assim que um dispositivo eletrônico como *smartphone*, adentrar ou sair desta área (WHITE, 2017).

A *geofence* pode ser configurada também pelo próprio usuário final, que se utilizando de aplicativos móveis que possuam o *geofencing* implementado, pode determinar uma certa área para que assim que estiver entrando ou saindo dela, o seu aplicativo ou *smartphone* possa executar alguma ação. O usuário pode por exemplo realizar uma pesquisa em algum aplicativo de vendas, e escolher o raio em que o aplicativo irá realizar essa pesquisa atrás de alguma loja que venda o produto buscado (WHITE, 2017).

Para se fazer uso de uma *geofence*, é necessário apenas que se tenha conexão com a *Internet*, seja por meio de *Wi-Fi* ou pelos dados móveis, não se necessita adquirir um *hardware* específico e nem do uso de *Bluetooth* (STATLER, 2016, p. 311).

Uma *geofence* é calculada utilizando-se da Fórmula de Haversine (STATLER, 2016, p. 314). A Fórmula de Haversine é utilizada para se calcular a distância geográfica na Terra entre duas coordenadas. Utilizando a latitude e longitude obtidas das duas coordenadas, é possível calcular a distância mais curta entre esses dois pontos através da fórmula retrocitada. A Fórmula de Haversine é dada por:

$$hav(\theta) = sen^2\left(\frac{\theta}{2}\right) = \frac{1 - cos(\theta)}{2} \tag{2}$$

$$hav\left(\frac{d}{r}\right) = hav(\varphi_2 - \varphi_1) + cos(\varphi_1)\, cos(\varphi_2)\, hav(\lambda_2 - \lambda_1) \tag{3}$$

Onde,

d: é a distância entre as duas coordenadas;

r: é o raio da esfera;

φ_1, φ_2: latitudes das coordenadas 1 e 2 respectivamente;

λ_1, λ_2: longitudes das coordenadas 1 e 2 respectivamente (MONAWAR; MAHMUD; HIRA, 2017).

As *geofences* não são sempre criadas de formas iguais, existem dois tipos principais de *geofences* que podem ser desenvolvidas, sendo que, cada tipo tem um custo, precisão, tempo e disponibilidade. *Geofences* podem ser ativas ou passivas (STATLER, 2016, p. 312).

As *geofences* ativas funcionam apenas quando o aplicativo em que foram implementadas está aberto, e geralmente possuem alguma dificuldade na sua implementação. Assim que o usuário abre o aplicativo, este irá inicializar o GPS para obter uma alta precisão da latitude e longitude de onde o usuário se encontra. Desta forma, o aplicativo saberá se o usuário se encontra dentro ou fora de suas *geofences*, que são mantidas dentro do próprio aplicativo ou armazenadas em algum servidor de forma remota (STATLER, 2016, p. 312).

O aplicativo saberá se o usuário está dentro ou fora de uma *geofence*, comparando a latitude e longitude do usuário com as latitudes e longitudes que estão armazenadas dentro do banco de dados ao qual o aplicativo está conectado. Esta alta precisão na determinação das coordenadas ao qual o usuário se encontra, é um fator no qual uma *geofence* passiva não se aplica. Através do uso do GPS, é possível até mesmo se utilizar de *geofences* muito menores, com o raio de até 10 metros, e possuir uma excelente precisão para detectar a geolocalização do usuário em questão (STATLER, 2016, p. 312).

Há um fator muito importante com relação a este tipo de implementação de *geofence* ativa: os usuários precisam estar com o aplicativo aberto em suas telas e estar sempre fazendo uso do aplicativo para que seja determinado se ele está fora ou dentro da *geofence*. Devido a este fator, a maioria das empresas, tendem a não gostar desse tipo de implementação. Em sua maioria, as empresas querem rastrear os locais que o usuário frequenta, ou a frequência com que visitam os seus concorrentes (STATLER, 2016, p. 312).

Uma grande parcela das soluções de *geofencing* encontradas em lojas de aplicativos, possuem apenas *geofencing* ativo. Para isso então, esses aplicativos ativam o modo de alta precisão do GPS apenas quando é aberto o aplicativo, e desativam assim que ele é encerrado, já que esse tipo de funcionamento do GPS demanda um alto uso da bateria do *smartphone* (STATLER, 2016, p. 312).

As *geofences* passivas também funcionam quando o aplicativo está aberto, a diferença para as *geofences* ativas, é que também funcionam mesmo quando os aplicativos estão minimizados, e até mesmo quando já foram encerrados. Este tipo de *geofence* apresenta uma maior dificuldade de implementação com relação à ativa, e não possui a mesma precisão, apesar de funcionar em segundo plano (STATLER, 2016, p. 313).

Esta *geofence*, apresenta uma grande vantagem sobre as *geofences* ativas, justamente por não precisar que o usuário tenha que abrir o aplicativo para só assim, saber se ele se encontra dentro ou fora da cerca virtual. Seu único ponto negativo então, é que ela não apresenta uma precisão tão grande quanto *geofences* ativas, e muito provavelmente, não irá funcionar em uma *geofence* menor do que 100 metros (STATLER, 2016, p. 313).

A diferença na precisão, se dá por conta da *geofence* passiva não fazer uso total do GPS, até porquê, isso iria reduzir drasticamente a bateria do *smartphone*, ao invés disso, ela utiliza muito pouco o GPS, além de utilizar uma combinação dos dados móveis do celular e dados de *Wi-Fi* (STATLER, 2016, p. 313).

Os sistemas operacionais de *iPhones* e *smartphones Android*, constantemente se conectam às torres de celular mais próximas e buscam por redes *Wi-Fi* que estejam disponíveis. As empresas que desenvolvem estes sistemas, possuem um banco de dados que contém a localização geográfica de cada torre de celular. Assim, sempre que um *smartphone* se conecta a uma dessas torres de celular, é possível ter uma aproximação de sua localização. A precisão, no entanto, será baixíssima, localizando o usuário em algum lugar dentro de um raio de centenas de metros, mas, assim que o *smartphone* se conectar a uma rede *Wi-Fi*, a precisão aumentará de acordo com os dados da mesma (STATLER, 2016, p. 313).

Aparelhos *iPhone* e *Android*, se utilizam das latitudes e longitudes que são marcadas nas redes *Wi-Fi* para terem uma maior precisão na localização do usuário. Sempre que estes aparelhos encontram uma nova rede que não se encontra no banco de dados de seus desenvolvedores, o sistema operacional aciona o GPS, marca as coordenadas e as envia de volta para os seus respectivos bancos de dados. Além disso, os *smartphones* não precisam estar conectados a uma rede *Wi-Fi*, bastando apenas deixarem o *Wi-Fi* ligado, que ao escanear o ambiente, já encontrará as redes *Wi-Fi* mais próximas, e obterá suas coordenadas (STATLER, 2016, p. 313).

Os aplicativos que fazem uso desse tipo de *geofence*, acabarão não funcionando de forma devida caso o *Wi-Fi* dos *smartphones* esteja desligado, necessitando sempre que o *smartphone* fique constantemente realizando o escaneamento da área para obter uma maior precisão de sua localização (STATLER, 2016, p. 313).

2.3 *Google Firebase*

O *Google Firebase* é uma plataforma *web* que foi desenvolvida pelo *Google* e lançada em 2016. Ela oferece diversas ferramentas e infraestrutura para se desenvolver aplicativos tanto para *Android*, como para *iOS* e aplicativos *web*. Dentre as diversas ferramentas e serviços, tem-se mensagens, banco de dados, autenticação de segurança e muitos outros (MORONEY, 2017, p. 1).

A plataforma do *Google Firebase* foi desenvolvida buscando auxiliar os desenvolvedores na construção de aplicativos de alta qualidade. Diferente de grande parte dos bancos de dados, o *Firebase* não utiliza uma tabela com linhas e colunas como a *Structured Query Language* (SQL) para armazenamento de dados. O *Firebase* utiliza o formato *JavaScript Object Notation* (JSON) para armazenar os dados, e que não faz uso de comandos conhecidos como *queries* para inserir, adicionar, deletar e atualizar os seus dados (KHAWAS; SHAH, 2018).

O *Firebase* oferece produtos e ferramentas para 3 tipos de serviços diferentes que são para criar aplicativos, para liberar e monitorar aplicativos buscando melhorar a qualidade, e para engajamento, visando aumentar os negócios de seus clientes. Muitos desses produtos estão disponíveis de forma gratuita como *Analytcs*, Indexação de Aplicativos, Autenticação, *Cloud Messaging*, Notificações e Configuração Remota. Outros produtos também são gratuitos, mas apenas para testes ou aplicativos menores, como no caso do *Realtime Database*, onde está disponível 1 GB de armazenamento de forma gratuita, mas se for preciso um maior volume para armazenar dados, terá de ser pago um valor a mais para se ter quantidade de armazenamento desejada. É possível escolher apenas o produto ou ferramenta que for ser utilizado, sem necessariamente ter que usar todas que estiverem disponíveis (MORONEY, 2017, p. 2).

A Figura 5 ilustra alguns dos serviços que estão disponíveis para serem utilizados.

Figura 5 – Opções de serviços na aba de Criação de Aplicativos e Engajamento

Fonte: Moroney, 2018.

A seguir, é apresentado alguns dos serviços e produtos utilizados para criação, desenvolvimento e engajamento de aplicativos:

2.3.1 Realtime Database

É um banco de dados baseado no modelo *Not Only Structured Query Language* (NoSQL), ou seja, que não faz uso de tabelas com linhas e colunas para armazenamento de dados, como já dito anteriormente. Mesmo sendo um banco de dados hospedado na nuvem, ele funciona de forma *off-line*, armazenando dados na memória do dispositivo utilizado, e atualizando estes dados assim que o dispositivo voltar a se conectar à *internet*. Este banco de dados funciona por meio regras que irão definir como os dados serão estruturados, e quais usuários terão acesso a estes dados, essas regras são conhecidas como regras de segurança (MORONEY, 2017, p. 3). Por fim, é fornecido uma *Application Programming Interface* (API), ou Interface de Programação de Aplicativo, que permite que os dados dos aplicativos sejam sincronizados e armazenados em nuvem (KHAWAS; SHAH, 2018).

2.3.2 Cloud Storage para Firebase

É um serviço de armazenamento de baixo custo que permite aos desenvolvedores e usuários realizarem *download* e *upload* de arquivos, como vídeos, fotos e áudio independente da qualidade da rede na qual o dispositivo está conectado (KHAWAS; SHAH, 2018). O *Cloud*

Storage possui suporte e integração com o *Google Cloud*, e realiza essas transferências de arquivos através do uso de uma API (MORONEY, 2017, p. 3).

2.3.3 Firebase Test Lab

O *Test Lab* é um serviço que oferece uma infraestrutura em nuvem para realização de testes dos aplicativos que estão sendo desenvolvidos. Esta ferramenta é utilizada por desenvolvedores para verificar se seus aplicativos irão funcionar corretamente em diversos dispositivos que possuam diferentes configurações. Ele funciona mesmo que os desenvolvedores não tenham programado um código para testagem, realizando testes próprios de maneira automática para verificar se o aplicativo em questão possui falhas, além de armazenar os resultados em vários formatos, como capturas de tela, no próprio *Firebase* (KHAWAS; SHAH, 2018).

Os dispositivos usados para realizar os testes estão disponíveis em um centro de testes do próprio *Google*, e os testes realizados de forma automática foram desenvolvidos pelo *Google* e são chamados de *Robo Tests* (MORONEY, 2017, p. 4).

2.3.4 Firebase Crash Reporting

O *Crash Reporting* é uma ferramenta que foi desenvolvida para ajudar a solucionar um grande problema que infelizmente diversos aplicativos apresentam, que são os erros e travamentos conhecidos com *crashes*. Essas falhas são o maior motivo de alguns aplicativos apresentarem baixas recomendações em suas lojas de aplicativos como *Google Play Store*. Essa ferramenta auxilia no processo de descobrir o motivo que levou um aplicativo a apresentar falhas. O *Crash Reporting* ajuda fornecendo um *stack trace* de todas as falhas que o aplicativo apresentar no console do *Firebase*. O *stack trace* se trata de uma lista das funções que estavam sendo executadas pelo aplicativo quando ocorreu uma exceção, sendo a exceção, uma forma do ambiente de programação indicar que ocorreu um erro. Após a ferramenta fornecer o *stack trace*, é possível utilizá-lo para descobrir qual a falha e assim implementar uma correção (MORONEY, 2017).

Os erros listados na *stack trace* são agrupados e mostrados de acordo com a gravidade, e o desenvolvedor pode manualmente implementar eventos personalizados para auxiliar na procura por *crashes* (KHAWAS; SHAH, 2018).

2.3.5 Firebase Cloud Messaging

O *Firebase Cloud Messaging* (FCM), é um serviço que permite aos desenvolvedores enviarem mensagens e notificações para os aplicativos clientes de forma gratuita e rápida. Essas mensagens podem ser enviadas de diversas formas, como utilizando análise e enviando para um público específico e outros métodos, podendo também complementar outras ferramentas do *Firebase* como auxiliando o *Crash Reporting* ao enviar mensagens para os usuários notificando de uma atualização que irá resolver algum erro que foi apresentado em um determinado aplicativo (MORONEY, 2017). É um serviço de *cross-platform* ou plataforma cruzada, ou seja, que irá agir em diversas plataformas como *Android*, *iOS* e aplicativos *web* (KHAWAS; SHAH, 2018).

2.3.6 Firebase Authentication

Diversos aplicativos hoje em dia requerem que seus usuários se cadastrem e criem credenciais de *login* para poderem fazer uso do mesmo, e devido a muitos desse aplicativos exigirem muitas informações pessoais no cadastro, podem levar seus usuários a não quererem se cadastrar e não fazerem uso do aplicativo em si. Por conta deste problema e da implementação de uma infraestrutura para criação e manutenção de credenciais de *login* ser algo difícil e custoso, o *Firebase Authentication* foi desenvolvido. Esta ferramenta oferece uma API que possibilita realizar cadastros e *logins* em aplicativos que façam uso dela, através de contas do *Google* e *Facebook* por exemplo, ou até mesmo realizar *logins* com apenas *e-mail* e senha. Aplicativos que apresentam este tipo de autenticação e permitem que o usuário faça *login* utilizando credenciais que ele já possui, tem se tornado cada vez mais populares. Esta ferramenta também pode ser integrada a outros serviços do *Firebase* como o *Realtime Database*, permitindo que haja um controle maior sobre quais usuários possuem acesso a certos tipos de dados (MORONEY, 2017).

2.3.7 Firebase Remote Config

O *Firebase Remote Config* é uma ferramenta de configuração remota que permite ao desenvolvedor realizar configurações e alterações tanto no comportamento quanto na aparência de seu aplicativo sem que necessite que ele receba uma nova atualização. Ela permite por

exemplo corrigir palavras que estejam mal traduzidas em algum idioma, ou controlar os descontos aplicados em alguns produtos dentro de um aplicativo de vendas (MORONEY, 2017).

2.3.8 Google Analytics para *Firebase*

O *Google Analytics* é uma ferramenta de engajamento tida como o núcleo do *Firebase*. É uma solução gratuita que reúne análises comuns e personalizadas sem a necessidade de desenvolver um código específico para isso. Ela possibilita ao desenvolvedor do aplicativo, conhecer e compreender seu público e como eles utilizam o seu aplicativo, assim, o desenvolvedor terá este auxilio para saber como prosseguir no crescimento e desenvolvimento deste aplicativo, trazendo melhorias de acordo com o que os usuários buscam ou precisam (KHAWAS; SHAH, 2018).

Esta ferramenta pode ser utilizada junto de outras como o *Remote Config*. Através das análises feitas, o desenvolvedor fornece uma opção extra em seu aplicativo para um determinado público, como por exemplo descontos em produtos para uma determinada região que não tenha muitos usuários deste aplicativo, visando a expansão naquele local, e alertando seus usuários através do *Cloud Messaging* (MORONEY, 2017).

2.4 *Android Studio*

O *Android Studio* é mais uma ferramenta desenvolvida pela *Google* e disponibilizada, a primeira versão estável, em 2014. Essa é uma ferramenta de ambiente de desenvolvimento integrado, o que permite um *Rapid Application Development* (RAD), ou um Rápido Desenvolvimento de Aplicação visando uma maior produtividade, essas ferramentas são caracterizadas por editar, compilar, associar, modelar dados, e gerar códigos (HELLMANN, 2016).

O *Android Studio* permite que o código seja reorganizado para *Android*, possui compatibilidade de refatoração com o *Android Wear*, um relógio com sistema operacional *Android*, logo é possível perceber que essa IDE é extremamente versátil facilitando a vida para o desenvolvedor. O *Android Studio* oferece também um ambiente para desenvolvimento robusto, todos os modelos e conceitos encontrados em toda programação *Android* e, por fim, é um recurso com muitas ferramentas extras (HELLMANN, 2016).

3 PROPOSTA DE SOLUÇÃO

Este capítulo tem por finalidade, apresentar a proposta de solução para este trabalho na qual será realizada no Trabalho de Conclusão de Curso II. Será apresentado as ferramentas que serão utilizadas para implementar o aplicativo *Android*, tendo como base os conceitos e definições apresentados no capítulo anterior.

3.1 *Firebase Realtime Database*

Este trabalho fará uso da plataforma *Google Firebase*, e utilizará a ferramenta *Realtime Database* apresentada anteriormente, para armazenar os dados utilizados pelo aplicativo *Android* a ser desenvolvido. O *Realtime Database* é um banco de dados hospedado na nuvem do tipo NoSQL, no qual os seus dados são armazenados no formato JSON. Por ser baseado em nuvem, sempre que os usuários estiverem conectados à rede, o banco de dados será sincronizado em tempo real, e receberão atualizações com os dados mais recentes. O *Realtime Database* permite apenas operações de rápida execução, justamente para possibilitar uma boa experiência com atualizações em tempo real (FIREBASE, 2020).

Por mais que os dados sejam armazenados em nuvem, o *Software Development Kit* (SDK), ou Kit de Desenvolvimento de *Software* do *Realtime Database*, permite que os dados do aplicativo também sejam salvos na memória do *smartphone*, mantendo os dados do aplicativo mesmo quando não houver conexão com a *Internet*, e atualizando os dados assim que a conexão é reestabelecida. A atualização é feita sincronizando as alterações realizadas pelos usuários com as atualizações disponibilizadas enquanto o *smartphone* estava *off-line*, mesclando os conflitos de dados automaticamente (FIREBASE, 2020).

A plataforma *Google Firebase* oferece dois planos de contratação para utilizar os serviços e ferramentas disponibilizados. O plano *Spark* é o plano gratuito, neste plano, o *Realtime Database* permite que seja utilizado apenas 1 banco de dados por projeto, 1 GB de armazenamento de dados, e no máximo 100 conexões simultânea ao banco de dados do aplicativo. O plano *Blaze* é um plano que cobrará taxa extras apenas em caso de extrapolar as quantidades gratuitas ofertadas no plano *Spark*, ou seja, o valor cobrado será referente ao excedente, aumentando seu valor de acordo com a quantidade extrapolada (BOHBOT, 2020).

Para o *Firebase* ser adicionado ao projeto de desenvolvimento do aplicativo *Android*, alguns requisitos são necessários. Primeiramente deve se ter instalado o programa *Android Studio* em sua versão mais recente. O aplicativo a ser desenvolvido necessita de possuir pelo menos nível 16 da API (*Jelly Bean*), além do *Gradle* 4.1, ou suas versões mais recentes. E por fim, se faz necessário o uso de um *smartphone* ou de um emulador para realizar a execução do aplicativo. A conexão do aplicativo *Android* ao *Firebase* pode ser feita de duas maneiras, utilizando o próprio Console do *Firebase*, que é a opção recomendada, ou utilizando o *Firebase* Assistente do *Android Studio*, e neste caso poderá exigir uma configuração adicional (FIREBASE, 2021).

3.2 *Android Studio*

Este aplicativo será desenvolvido através do ambiente de desenvolvimento *Android Studio*. O *Android Studio* é a IDE oficial criada pelo *Google* para desenvolver aplicativos para *Android*. Ela oferece diversos recursos além do editor de códigos e das ferramentas de desenvolvimento avançado. As ferramentas disponíveis incluem emulador de dispositivos *Android* de alto desempenho, ambiente de desenvolvimento unificado que possibilita a criação de aplicativos para todos os dispositivos *Android*, alterações no código enquanto o aplicativo está executando, sem a necessidade de reiniciar o mesmo, ferramentas de teste e ferramentas para detecção de problemas de desempenho, usabilidade e compatibilidade, entre outras (DEVELOPERS, 2021).

O projeto de desenvolvimento é estruturado em módulos, nos quais podem ser vistos no nível *Gradle Scripts*. Cada um dos módulos possui 3 pastas. A pasta "*manifest*" contém o arquivo "*AndroidManifest.xml*", a pasta "*Java*" possui os arquivos de código-fonte do *Java*, e a pasta "Recursos" possui todos os recursos do aplicativo que não são códigos, incluindo *layouts eXtensible Markup Language* (XML) e imagens em *bitmap* (DEVELOPERS, 2021).

O *Android Studio* utiliza o sistema de compilação *Gradle*, que é uma ferramenta integrada ao menu da plataforma de desenvolvimento e que trabalha de maneira independente da linha de comando. Através dos recursos deste sistema de compilação, é possível ampliar, personalizar e configurar o processo de programação, além de possibilitar a criação de diversos *Android Aplication Package* (APK) para o aplicativo desenvolvido, podendo cada um possuir diferentes recursos, mesmo utilizando o mesmo projeto. Este sistema possibilita a criação de diversas variações de um mesmo aplicativo se utilizando de apenas um projeto, permitindo criar aplicativos com versões pagas e gratuitas e aplicativos com recursos diferentes que funcionarão

de acordo com a configuração do dispositivo no qual o aplicativo será executado (DEVELOPERS, 2021). Este trabalho será desenvolvido utilizando a linguagem de programação *Java*.

3.3 *Android* SDK

Neste projeto, também será utilizado o kit de desenvolvimento *Android* na criação do aplicativo. O *Android SDK* possui diversas ferramentas de desenvolvimento, amostras de códigos-fonte, bibliotecas necessárias para o desenvolvimento de aplicativos, e um emulador utilizado para simular um dispositivo *Android*, permitindo a execução neste emulador para realização de testes antes de disponibilizar o aplicativo desenvolvido para *download*. Este SDK também fornece um extenso conjunto de APIs para implementar no aplicativo a ser desenvolvido (GOLHAR, *et. al.*, 2016).

Os aplicativos desenvolvidos são executados nas máquinas virtuais personalizadas *Android Runtime* (ART) e *Dalvik*, projetadas para serem executadas em dispositivos *Android* que funcionem em um *kernel Linux*. Sempre que uma nova versão do sistema *Android* é lançado pelo *Google*, um SDK correspondente a esta versão também é lançado, fazendo-se necessário a instalação deste novo SDK para poder prosseguir com o desenvolvimento do aplicativo para a nova versão do sistema operacional. O uso do SDK se faz através da integração com a IDE *Android Studio*, que possibilita o uso das ferramentas fornecidas pelo SDK instalado (CORDEIRO, 2018).

3.4 Aplicativo móvel

Esta seção tem por finalidade, apresentar todos componentes implementados no código-fonte do aplicativo *Android*, incluindo métodos e funcionalidades desenvolvidas para o mesmo.

3.4.1 Maps Activity

A *Maps Activity* se trata da interface principal deste aplicativo, a qual irá aparecer primeiramente quando executar o mesmo. Ela contém as funcionalidades e serviços necessários a serem utilizados na implementação da *geofencing* por meio do *Google Maps*. Para que o SDK do *Maps* para *Android* funcione de fato, é necessário inserir uma chave de API, onde a mesma

deve ser obtida por meio do registro na plataforma *Google Cloud*, que possui uma biblioteca de APIs, incluindo o SDK do *Maps* para *Android* (DEVELOPERS, 2021b).

Ao se criar um projeto no *Android* Studio e escolher a opção *Google Maps Activity*, serão gerados três arquivos:

- *MapsActivity.java*: este arquivo contém os métodos e funcionalidades que permitirá que sejam realizadas manipulações no mapa, possibilitando também, o controle de ações por parte do usuário (DEVELOPERS, 2021b).
- *activity_maps.xml*: este arquivo contém os elementos de interface da tela principal do aplicativo, como uma *SeekBar*, que o usuário utiliza para definir o tamanho do raio da *geofence*, e um mostrador do tamanho deste raio em metros. O arquivo também contém o *fragment* do *Google Maps*. Um *fragment* se trata de uma parte da interface de usuário dentro de uma *Activity*, sendo considerado uma seção modular da mesma, no qual possui o próprio ciclo de vida e próprios comportamentos. Ao executar o aplicativo, o *fragment* será instanciado, e irá gerar o mapa do *Google* automaticamente (DEVELOPERS, 2019).
- *google_maps_api.xml*: este arquivo serve para armazenar a chave de API que será utilizada neste aplicativo (DEVELOPERS, 2021b).

Para que o aplicativo tenha acesso a localização do usuário, será necessário inserir as seguintes permissões no arquivo *AndroidManifest.xml*:

- *android.permission.ACCESS_FINE_LOCATION*: ao ser aceita, esta permissão fará com que o aplicativo possa fazer uso do receptor GPS do *smartphone* juntamente com o *Wi-Fi* e os dados móveis, possibilitando uma precisão muito maior, geralmente de 50 metros, e até mesmo de 3 metros (DEVELOPERS, 2021c, 2021d).
- *android.permission.ACCESS_COARSE_LOCATION*: esta permissão também possibilitará que o aplicativo tenha acesso a localização do usuário, mas neste caso, fará uso apenas dos dados móveis e *Wi-Fi*, e assim, terá uma precisão inferior, com uma estimativa de cerca de 1,6 km (DEVELOPERS, 2021c, 2021d).

No arquivo *MapsActivity.java* será realizada a implementação do *fragment* do *Google Maps*, no qual possui alguns métodos que são sobrescritos, e para isso é necessário utilizar a diretiva de compilação *override*. Os métodos sobrescritos são *onCreate*(), *onCreateOptionsMenu*(), *onOptionsItemSelected*(), *onResume*(), *onMapReady*(), e

onResquestPermissionResult(), além desses, também são instanciados métodos para uso do próprio aplicativo como *permissionsAsk*() e *geofenceMarker*().

O método *onCreate*() é chamado assim que o aplicativo é aberto. Dentro dele será feita a solicitação de permissão de localização (caso seja a primeira vez que o aplicativo é executado), logo após isso, ele lerá os dados que foram previamente inseridos no banco de dados do *Firebase*, inserindo-os em suas devidas variáveis. Irá também inicializar algumas outras variáveis, além de inicializar o próprio mapa.

Para representar as lojas e seus respectivos produtos, foi simulado um banco de dados contendo no total 5 lojas, com cada uma contendo sua latitude, longitude, nome e uma lista com o total de 6 produtos, que por sua vez, possuindo sua marca, modelo, preço, característica específica do produto e uma referência para sua imagem armazenada no *Cloud Storage* do *Firebase*.

A Figura 6 mostra a organização das lojas no banco de dados *Realtime Database*, e a Figura 7 apresenta a organização dos produtos em cada loja.

Figura 6 – Disposição dos dados das lojas inseridos no *Realtime Database*

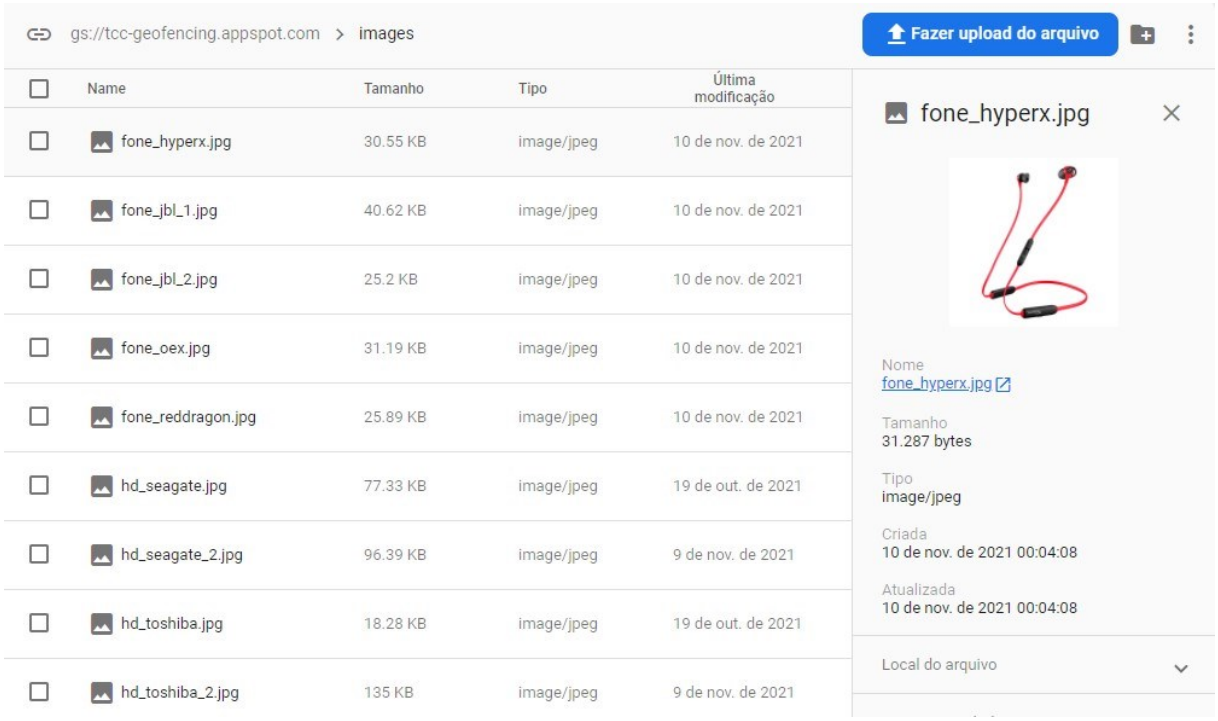
Fonte: Elaborado pelo autor.

Figura 7 – Disposição dos dados dos produtos inseridos no *Realtime Database*

Fonte: Elaborado pelo autor.

A Figura 8 exibe a tela do *Cloud Storage* do *Firebase*, onde foram inseridas as imagens de cada produto.

Figura 8 – Imagens armazenadas no *Cloud Storage* do *Firebase*



Fonte: Elaborado pelo autor.

No método *onResume*() que ocorrerá o recebimento da localização de fato. Neste método, será chamado o método *getLastLocation*(), que irá retornar a última localização em que o *smartphone* esteve, após isso, utilizando a classe *LocationRequest*. é definido o intervalo com que o aplicativo solicitará a localização do usuário, de forma que, quanto menor o intervalo, mais é exigido do dispositivo GPS, e com isso, o consumo da bateria aumentará, isso não é aconselhado para um dispositivo móvel como *smartphones*.

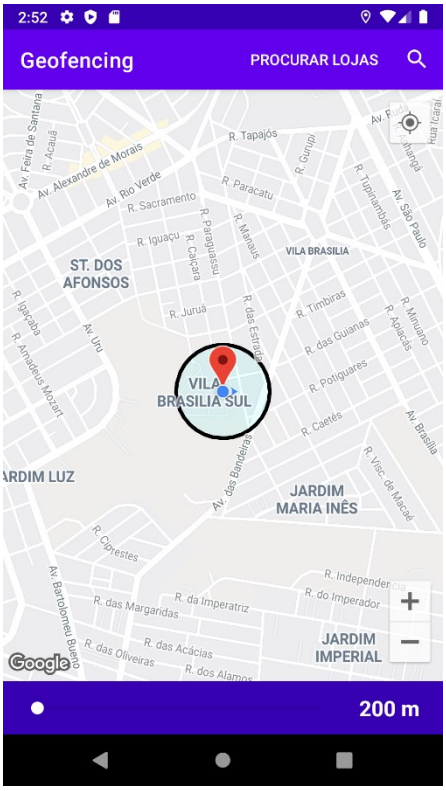
Também é definido a prioridade deste aplicativo, neste definida como balanceada, ou seja, possuirá uma boa precisão, de até 100 metros, mas ainda assim, conseguirá manter uma melhor economia da bateria. Logo após obter a localização atual do usuário, também é inserido um marcador em sua localização, juntamente de um círculo, representando graficamente uma *geofence* (DEVELOPERS, 2021f).

O método *onMapReady*() será chamado assim que o mapa inicializado no método *onCreate*() estiver pronto. Neste método são realizadas as configurações do mapa, no qual é inicializado já com a localização do usuário, e também é definido o zoom máximo como 20f e o zoom mínimo como 5f, sendo que o valor 1f corresponde o nível de zoom de mundo, 5f de continentes, 10f de cidades, 15f de ruas, e por fim, 20f de construções. Neste método, também é disponibilizado para o usuário final, meios de manipular o mapa como utilizando de gestos para diminuir ou aumentar o zoom, controle do zoom por meio de botões, inclinação e rotação do mapa, além de adicionar marcadores geográficos por meio de um clique longo na tela do *smartphone*, também permite o usuário a aumentar ou diminuir o tamanho da *geofence* através de uma *SeekBar* inserida logo abaixo do mapa (DEVELOPERS, 2021e).

O método *onRequestPemissionsResult*() é chamado para verificar se as permissões solicitadas na primeira vez em que o aplicativo é executado, foram concedidas, e caso alguma delas tenha sido negada, aparecerá uma caixa de texto informando que para o aplicativo funcionar da maneira correta, se faz necessário que o usuário aceite as permissões solicitadas, e após isso, o aplicativo é fechado, para que quando o usuário abri-lo novamente, ele aceite as permissões.

Após a implementação de todas as funcionalidades e componentes descritos acima, a interface inicial do aplicativo é apresentada conforme a Figura 9.

Figura 9 – Interface inicial do aplicativo

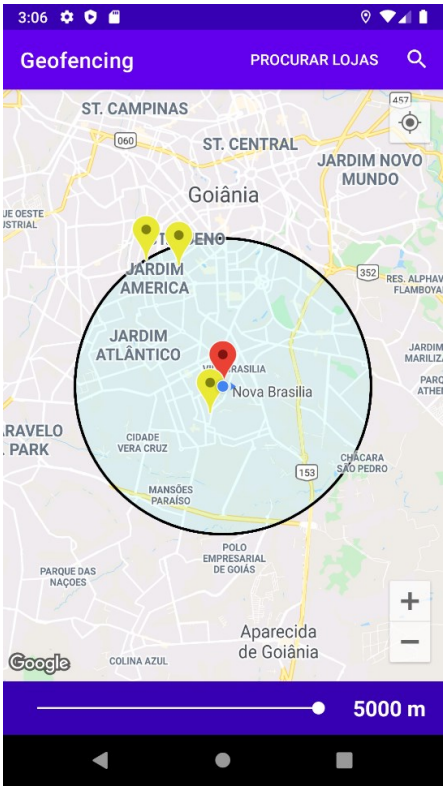


Fonte: Elaborado pelo autor.

O método *onCreateOptionsMenu*() irá inicializar uma barra de menu de opções chamada de *Toolbar* na parte superior da interface principal, cuja interface foi criada dentro do arquivo *menu_procurar_lojas.xml*, na qual possuirá diversas opções que funcionarão como um botão.

Já o método *onOptionsItemSelected*(), funcionará em conjunto do método *onCreateOptionsMenu*(). Este método irá implementar quais ações serão executadas para cada opção escolhida na *Toolbar*. Primeiramente foi implementada a opção “Procurar Lojas”. Ao clicar nesta opção, o método em questão irá realizar uma verificação na área do mapa coberta pela *geofence* definida pelo usuário, atrás de lojas que estejam inseridas no banco de dados, e apresentá-las ao usuário na forma de marcadores amarelos caso encontre uma ou mais lojas, conforme exibido na Figura 10.

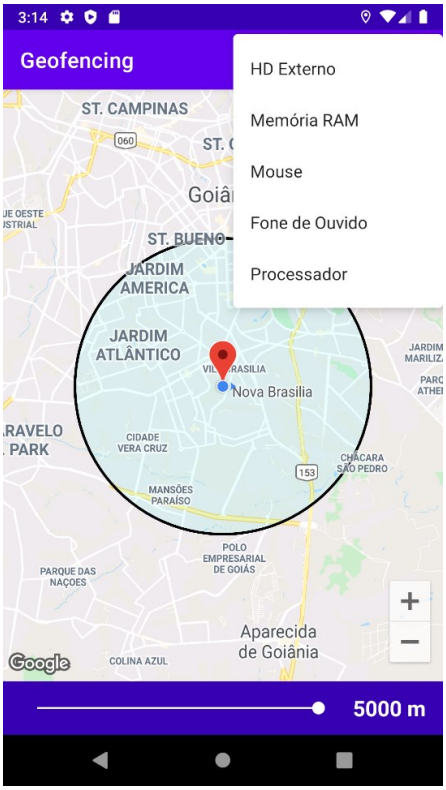
Figura 10 – Interface do aplicativo ao encontrar lojas



Fonte: Elaborado pelo autor.

Após isso, foi implementado um menu de itens com o ícone de uma Lupa, que, ao ser clicado, irá abrir um leque de opções de nomes de produtos a serem buscados. As opções incluídas são “HD Externo”, “Memória RAM”, “Mouse”, “Fone de Ouvido” e “Processador”, e são exibidas de acordo com a Figura 11.

Figura 11 – Menu de opções de busca de produtos



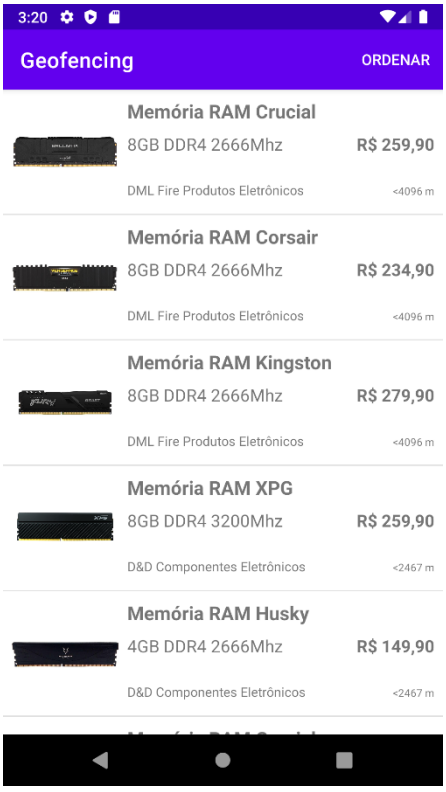
Fonte: Elaborado pelo autor.

Ao clicar em qualquer uma destas opções, o método fará uma busca na região para verificar se existe alguma loja dentro do raio da *geofence*, e caso encontre uma ou mais lojas, será aberta uma nova *Activity* com o nome *ProdutosLoja Activity*, que exibirá uma lista do produto selecionado. As Figuras 12, 13, 14, 15 e 16 exibem a lista de cada produto apresentado anteriormente.

Figura 12 – Interface com a lista de HD Externo

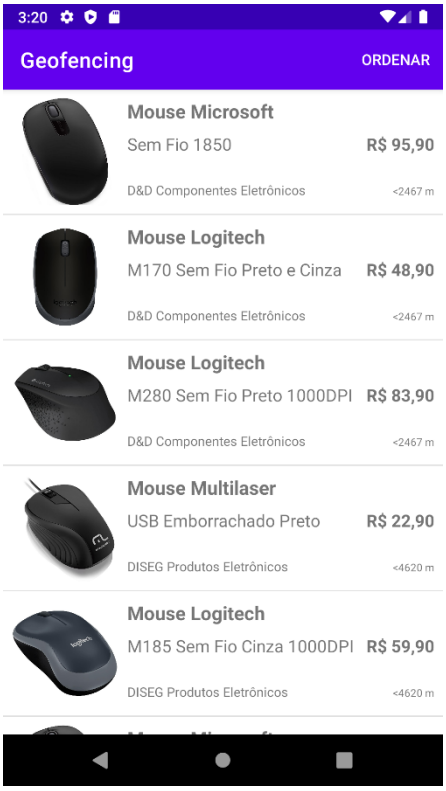
Fonte: Elaborado pelo autor.

Figura 13 – Interface com a lista de Memória RAM



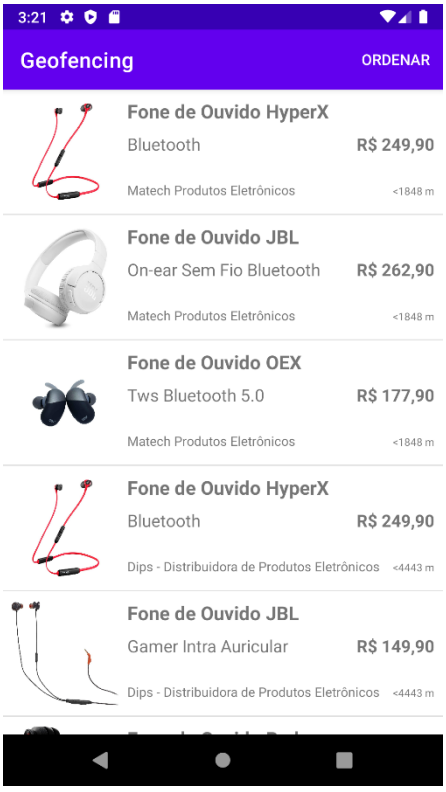
Fonte: Elaborado pelo autor.

Figura 14 – Interface com a lista de Mouse



Fonte: Elaborado pelo autor.

Figura 15 – Interface com a lista de Fone de Ouvido



Fonte: Elaborado pelo autor.

Figura 16 – Interface com a lista de Processador

Fonte: Elaborado pelo autor.

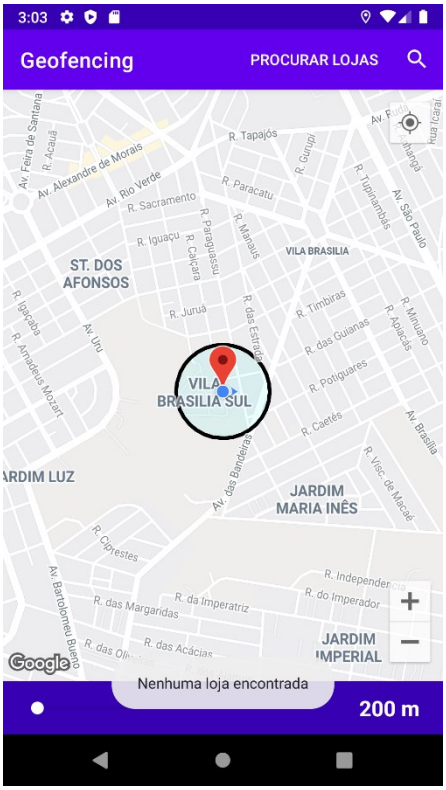
Caso encontre uma ou mais lojas, mas estas não possuam o item escolhido, será exibida uma tela informando de que nenhum produto foi encontrado, como apresentada na Figura 17.

Figura 17 – Interface do aplicativo ao não encontrar o produto escolhido

Fonte: Elaborado pelo autor.

E por fim, no caso de não encontrar alguma loja, será exibida uma mensagem na forma de uma visualização flutuante chamada de *Toast*, informando ao usuário sobre esta questão, conforme apresentado pela Figura 18.

Figura 18 – Interface do aplicativo ao não encontrar alguma loja



Fonte: Elaborado pelo autor.

Dentro do método *onMapReady*(), juntamente do método *onOptionsItemSelected*(), caso uma ou mais lojas sejam encontradas ao realizar a busca, será exibido no mapa todas as lojas encontradas na forma de marcadores amarelos, os quais possuirão como informações o nome da loja, e a distância que ela está do centro da *geofence*. Ao clicar nos marcadores das lojas, será exibido além das informações contidas em cada marcador, mencionadas anteriormente, um botão no canto superior esquerdo da tela com o título “Ver Produtos”, que, ao ser clicado, também irá abrir a *ProdutosLoja Activity*, mas neste caso, irá exibir uma lista contendo os produtos que estão dentro da loja selecionada anteriormente.

Ao clicar em uma das lojas marcadas no mapa, a interface do aplicativo será apresentada conforme a Figura 19.

Figura 19 – Interface do aplicativo após clicar em uma loja

Fonte: Elaborado pelo autor.

3.4.2 ProdutosLoja Activity

A *ProdutosLoja Activity* é a segunda interface implementada neste aplicativo, e ela possuirá dois arquivos, sendo eles o *ProdutosLojaActivity.java* e *activity_produtos_loja.xml*. Esta *Activity* contém as funcionalidades necessárias para exibir os produtos buscados pelo usuário na forma de uma lista vertical, seja todos os produtos de uma loja só, ou um tipo de produto de diversas lojas.

O arquivo *activity_produtos_loja.xml* contém os elementos de interface desta *Activity*, neste caso, possuindo apenas um *RecyclerView*. O *RecyclerView* cria uma lista de itens a serem exibidos para o usuário, neste caso, os itens serão os produtos das lojas, e como indicado pelo próprio nome, ele recicla os elementos exibidos, de forma que quando um elemento é rolado para fora da exibição, sua visualização não é destruída. Ao invés disso, o *RecyclerView* irá reutilizar esta visualização para exibir os novos elementos que forem aparecendo na tela conforme o usuário for rolando a lista, de forma a melhorar o desempenho do aplicativo, aumentando a capacidade de resposta e diminuindo o consumo energético (DEVELOPERS, 2021g).

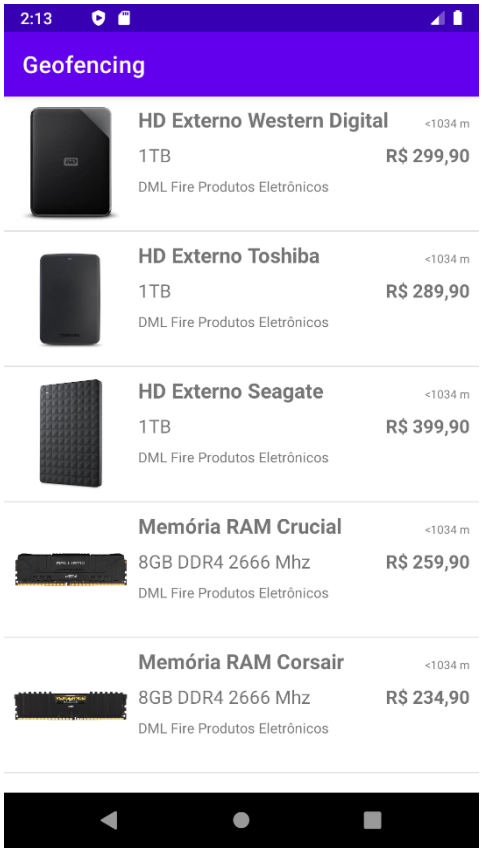
O arquivo *ProdutosLojaActivity.java* contém a implementação do *RecyclerView* mencionado anteriormente. A implementação está contida no método sobrescrito *onCreate*(). Assim que esta *Activity* é criada, este método é chamado. Ele irá primeiramente receber os dados enviados pela *Maps Activity* e inseri-los em suas devidas variáveis. Após isso, ele irá instanciar a classe *Adapter* passando as variáveis mencionadas anteriormente como parâmetro e fará a configuração do *RecyclerView*.

A classe *Adapter* está implementada no arquivo *Adapter.java*, o qual funciona em conjunto com o arquivo *adapter_lista.xml*. No arquivo *Adapter.java* está contida a implementação da classe em questão, possuindo as funcionalidades que irão inicializar cada item a ser exibido na lista do *RecyclerView*. O uso desta classe se faz necessário pois o *RecyclerView* precisa que seja definido um adaptador que associe os dados de cada item às visualizações que serão criadas pelo *RecyclerView* (DEVELOPERS, 2021g).

O arquivo *adapter_lista.xml* contém o *layout* do formato no qual cada item será exibido na lista do *RecyclerView*. Para cada item da lista, será exibido o modelo e a marca do produto, seu preço, o nome da loja na qual aquele determinado produto pertence, uma especificação do produto como a capacidade de armazenamento de um HD externo, a distância que a loja está do centro da *geofence*, e por fim, uma imagem do produto. O *RecyclerView* então, irá replicar este *layout* para cada produto que for sendo exibido, de forma a criar uma lista de visualizações verticais.

Após a implementação das funcionalidades descritas, a interface do aplicativo contendo a lista de produtos será apresentada conforme a Figura 20.

Figura 20 – Interface do aplicativo contendo a lista de produtos

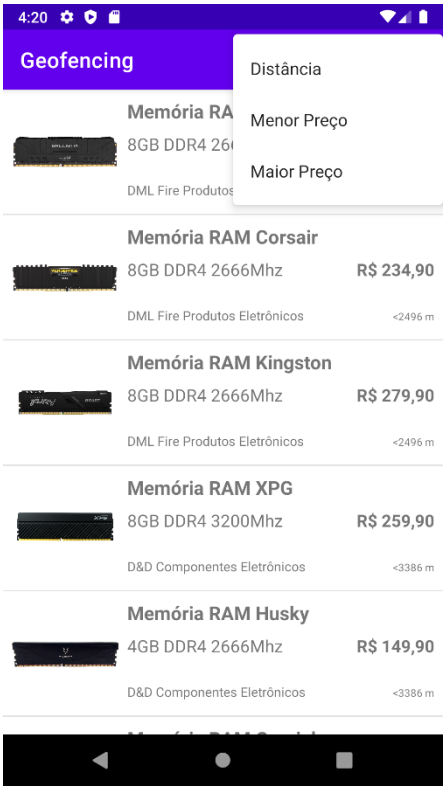


Fonte: Elaborado pelo autor.

Assim como na *MapsActivity*, esta também possuirá uma *Toolbar*, e fará uso dos métodos sobrescritos *onCreateOptionsMenu*() e *onOptionsItemSelected*().

O método *onCreateOptionsMenu*() irá inicializar a *Toolbar* contendo um menu de itens com o título “Ordenar”, que por sua vez, irá conter três itens a serem selecionados, cujos títulos são “Distância”, “Menor Preço” e “Maior Preço”, que serão exibidos ao clicar na opção “Ordenar” conforme exibido na Figura 21.

Figura 21 – Interface com as opções de ordenação



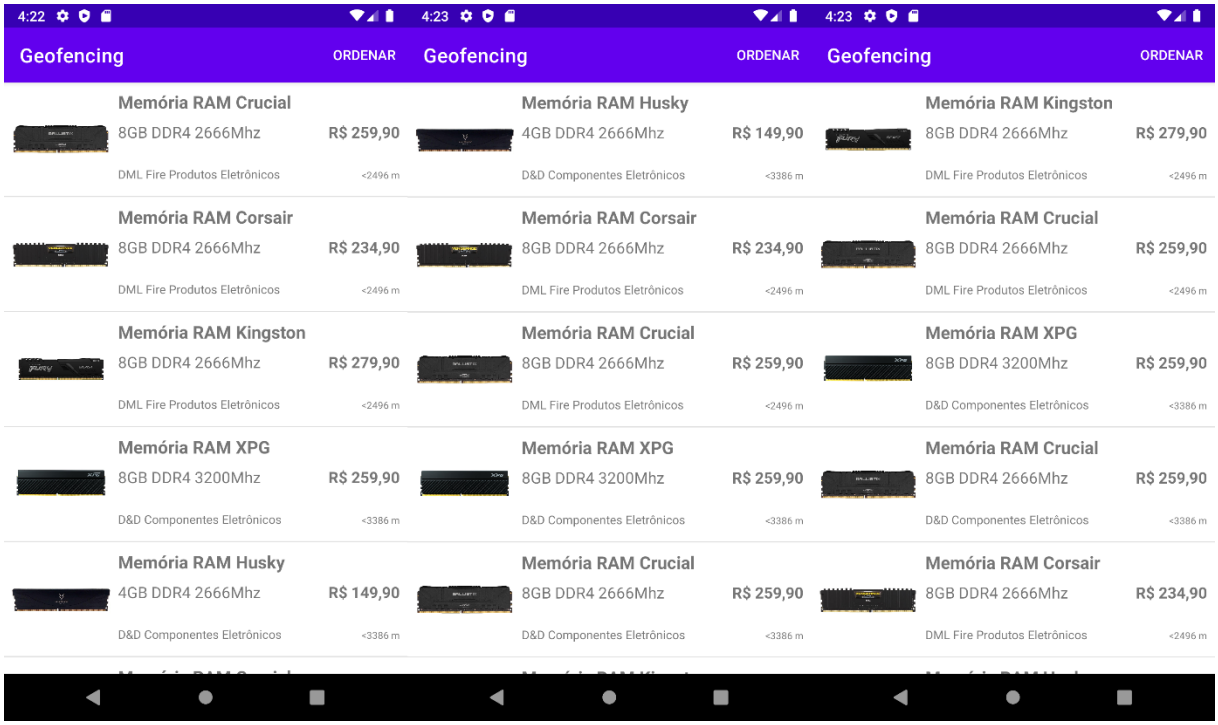
Fonte: Elaborado pelo autor.

Estes itens serão inicializados apenas se o usuário tiver selecionada uma busca por um produto específico, caso o usuário clique no botão “Ver Produtos” para visualizar os produtos de uma determinada loja, o menu de itens não será exibido.

O método *onOptionsItemSelected*(), irá chamar outros métodos de acordo com item selecionado. Estes métodos farão uma ordenação, utilizando da técnica *BubbleSort*, na lista de produtos a ser exibida conforme a opção selecionada, sendo elas, ordenar por menor distância, pelo menor preço e por fim, pelo maior preço dos produtos.

As ordenações por distância, menor preço e maior preço serão exibidas de acordo com a Figura 22.

Figura 22 – Lista ordenada por Distância, Menor Preço e Maior Preço, respectivamente



Fonte: Elaborado pelo autor.

Todos os códigos desenvolvidos estão apresentados nos Apêndices de A até J no final do presente trabalho.

4. CONSIDERAÇÕES FINAIS E PERSPECTIVAS FUTURAS

O presente trabalho teve por finalidade, a criação de um aplicativo de buscas por produtos eletrônicos dentro de uma determinada área no globo terrestre denominada *geofence*.

O IDE *Android* Studio possibilitou o desenvolvimento do aplicativo por meio da linguagem de programação *Java*, mostrando-se uma excelente ferramenta, possuindo diversas bibliotecas contendo milhares de métodos e classes. Sua facilidade com a integração das APIs e SDKs surpreendeu bastante, visto que são necessários poucos passos para realizar a integração, onde os mesmos são ensinados pelo próprio *Google*. Por meio da comunidade de desenvolvedores *Android* tanto do *Google* quanto de *sites* de terceiros, foi possível tirar diversas dúvidas de forma precisa e seguir com a implementação.

O *Firebase* é uma ferramenta de grande ajuda, permitindo a simulação dos dados das lojas e dos produtos, e possibilitando a recuperação e uso dos mesmos em tempo real. Sua plataforma possui uma gama extensa de documentação de fácil entendimento que nos auxilia na implementação. A configuração do *Realtime Database*, tanto das regras de leitura e escrita quanto da inserção dos dados, foi feita de forma facilitada na página do *Firebase Console*, onde também foi realizado o upload das imagens dos produtos armazenados no *Cloud Storage*. O plano de uso *Spark*, que é o plano gratuito do *Firebase*, atendeu bem as necessidades deste projeto.

O projeto proporcionou um grande aumento do conhecimento em várias áreas como desenvolvimento para *Android* com *Java*, utilização do SDK do *Firebase* e suas ferramentas, utilização da API *Maps* da *Google* e a criação de banco de dados em nuvem, além de todo conhecimento que foi utilizado como base teórica para o desenvolvimento do aplicativo. Além disso, este projeto apresenta um grande potencial para ser continuado, possibilitando a adição de novas funcionalidades. Os objetivos propostos neste projeto foram todos cumpridos.

Como projetos futuros sugere-se implementar um mecanismo de busca dos produtos por palavras, um sistema de filtragem de características dos produtos após realizar a busca, implementar uma tela que se abra ao clicar em algum produto, mostrando mais imagens do mesmo e suas características, implementar uma busca por produtos em lojas virtuais e não apenas físicas, implementar uma busca por lojas e produtos utilizando-se de mais de uma *geofence* ao mesmo tempo, infelizmente não houve tempo para realizar a implementação destas mesmas funcionalidades.

REFERÊNCIAS

BOHBOT, Kobi. **Preços do *Google Firebase* Para Leigos**, 2020. Disponível em: https://blog.back4app.com/pt/precos-do-google-firebase-para-leigos/. Acesso em: 24 maio 2021.

BRANDÃO, Bruna. Como funciona a Geolocalização na prática? Para que serve? **Maplink**, 2019. Disponível em: https://maplink.global/blog/como-funciona-geolocalizacao/#:~:text=Geolocalização%20é%20um%20recurso%20que,em%20um%20sistema%20de%20coordenadas. Acesso em: 24 mar 2021.

CORDEIRO, Fillipe. ***Android* SDK: O que é? Para que Serve? Como Usar?**, 2018. Disponível em: https://www.androidpro.com.br/blog/android-studio/android-sdk/. Acesso em: 24 maio 2021.

DEMPSEY, Caitlin. *Global Navigation Satellite Systems.* ***GIS Lounge***, 2011. Disponível em: https://www.gislounge.com/global-navigation-satellite-systems/. Acesso em: 12 abr 2021.

DEVELOPERS. **Fragmentos**. 2019. Disponível em: https://developer.android.com/guide/components/fragments?hl=pt-br. Acesso em: 28 set 2021

DEVELOPERS. **Conheça o *Android Studio***, 2021a. Disponível em: https://developer.android.com/studio/intro?hl=pt-br. Acesso em: 24 maio 2021.

DEVELOPERS. **Guia de início rápido do SDK do *Maps* para *Android***, 2021b. Disponível em: https://developers.google.com/maps/documentation/android-sdk/start. Acesso em 27 set 2021.

DEVELOPERS. **Solicitar permissões de localização**, 2021c. Disponível em: https://developer.android.com/training/location/permissions?hl=pt-br. Acesso em: 28 set 2021.

DEVELOPERS. **Dados de local**, 2021d. Disponível em: https://developers.google.com/maps/documentation/android-sdk/location?hl=pt-br. Acesso em: 28 set 2021.

DEVELOPERS. **Câmera e visualização**, 2021e. Disponível em: https://developers.google.com/maps/documentation/android-sdk/views?hl=pt-br. Acesso em: 28 set 2021.

DEVELOPERS. ***LocationRequest***, 2021f. Disponível em: https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest. Acesso em: 28 set 2021.

DEVELOPERS. **Criar listas dinâmicas com o *RecyclerView***, 2021g. Disponível em: https://developer.android.com/guide/topics/ui/layout/recyclerview?hl=pt-br. Acesso em: 25 out 2021.

FAGGIAN, Hugo César. **Geometria e GPS**. 2019. 60 f. Dissertação (Mestrado) - Curso de Matemática em Rede Nacional, Instituto de Ciências Matemáticas e de Computação ICMCUSP, Universidade de São Paulo (USP), São Carlos, 2019. Disponível em:

https://www.teses.usp.br/teses/disponiveis/55/55136/tde-17092019-150542/publico/HugoCesarFraggian_revisada.pdf. Acesso em: 24 mar 2021.

FIREBASE. **Adicionar o *Firebase* ao projeto para *Android***, 2021. Disponível em: https://firebase.google.com/docs/android/setup?hl=pt-br. Acesso em: 24 maio 2021.

FIREBASE. ***Firebase Realtime Database***, 2020b. Disponível em: https://firebase.google.com/docs/database?authuser=0. Acesso em: 24 maio 2021.

GOLHAR, Reetesh V.; VYAWAHARE, Prasann A.; BORGHARE, Pavan H.; MANUSMARE, Ashwini. *Design And Implementation Of Android Base Mobile App For An Institute. In: 2016 INTERNATIONAL CONFERENCE ON ELECTRICAL, ELECTRONICS, AND OPTIMIZATION TECHNIQUES (ICEEOT)*, 1., 2016, Chennai, India. **Artigo**. [S.L.]: IEEE, 2016. p. 3660-3663.

KHAWAS, Chunnu; SHAH, Pritam. *Application of Firebase in Android App Development-A Study*. ***International Journal of Computer Applications***. Nova Iorque, EUA, p. 49-53. jun 2018.

HELLMANN, Rafael Alexandre Freiberger. **Aplicativo *Android* e *website* interativos para busca de menores preços de produtos com código de barras**. 2016. 90 f. TCC (Graduação) - Curso de Engenharia Elétrica, Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2016.

MONAWAR, Tareq; MAHMUD, Shafayat Bin; HIRA, Avijit. *Anti-Theft Vehicle Tracking And Regaining System With Automatic Police Notifying Using Haversine Formula*. *In*: *INTERNATIONAL CONFERENCE ON ADVANCES IN ELECTRICAL ENGINEERING* (ICAEE), 4., 2017. Dhaka, Bangladesh. **Artigo.** [S.L.]: IEEE, 2017. p. 775-779.

MORONEY, Laurence. *An Introduction to Firebase*. *In*: MORONEY, Laurence. ***The Definitive Guide to Firebase****: Build Android Apps on Google's Mobile Platform*. Seattle, EUA: Apress, 2017. Cap. 1. p. 1-24.

POLON, Luana. Coordenadas Geográficas. **Estudo Prático**, 2018. Disponível em: https://www.estudopratico.com.br/coordenadas-geograficas/. Acesso em: 14 abr 2021.

PRINA, Bruno Zucuni; TRENTIN, Romario. Cálculo de Área no Sistema Geodésico Local: Georreferenciamento de Imóveis Rurais/Brasil. **Revista Continentes**, [S.l.], n. 11, p. 127-143, abr 2018. Disponível em: http://www.revistacontinentes.com.br/index.php/continentes/article/view/176. Acesso em: 14 abr 2021.

SANTANA, John Kennedy Ribeiro de; FARIAS, Paulo Lucas Cândido de; XAVIER, Joaquim Pedro de Santana; FIGUEIREDO, Victor Pina. Precisão de GPS em *smartphones*: Uma ferramenta para pesquisas acadêmicas e trabalhos de campo. **Revista de Geografia - PPGEO - UFJF**, [S.L.], v. 9, n. 2, p. 255-267, 06 abr 2020.

STATLER, Stephen. *Geofencing: Everything You Need to Know*. *In*: STATLER, Stephen. ***Beacon Technologies****: The Hitchhiker's Guide to the Beacosystem.* San Diego, EUA: Apress, 2016. Cap. 17. p. 307-316.

TEMBHEKAR, Trupti Deoram; SAKHARE, Trupti Jayant. *Informative Ideas to Describe Some Aspect of Latitude & Longitude Which Involved in Geographic Coordinate System.* ***Journal Of University Of Shanghai For Science And Technology***. Xangai, China, p. 30-33. jan 2021.

WHITE, Sarah K. ***What is geofencing? Putting location to work***. CIO Magazine, IDG Communications, Inc. 01 nov. 2017. Disponível em: https://www.cio.com/article/2383123/geofencing-explained.html. Acesso em: 08 mar. 2021.

APÊNDICE A – Código MapsActivity.java

```
package com.tcc.geofencing.activity;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import android.Manifest;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentSender;
import android.content.pm.PackageManager;
import android.content.res.Resources;
import android.graphics.Color;
import android.location.Geocoder;
import android.location.Location;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.gms.common.api.ResolvableApiException;
import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationCallback;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationResult;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.location.LocationSettingsRequest;
import com.google.android.gms.location.LocationSettingsResponse;
import com.google.android.gms.location.SettingsClient;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.Circle;
import com.google.android.gms.maps.model.CircleOptions;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MapStyleOptions;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.firebase.database.DataSnapshot;
```

```
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.tcc.geofencing.Buscador;
import com.tcc.geofencing.Loja;
import com.tcc.geofencing.Produto;
import com.tcc.geofencing.R;
import com.tcc.geofencing.databinding.ActivityMapsBinding;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class MapsActivity extends AppCompatActivity implements OnMapReadyCallback {

    private FusedLocationProviderClient client;
    private GoogleMap mMap;
    private Marker marker;
    private Circle circle;
    private LatLng home;
    private Button buttonProdutos;
    private ActivityMapsBinding binding;
    private DatabaseReference reference = FirebaseDatabase.getInstance().getReference();
    private List<Loja> loja = new ArrayList<Loja>();
    private String[] permissions = new String[]{
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION,
    };
    private SeekBar seekBar;
    private boolean firstTime = true;
    private TextView textSize;
    private int countLoja;
    private List<Marker> markerLoja = new ArrayList<Marker>();
    private List<Buscador> buscador = new ArrayList<Buscador>();
    int positionLoja;
    int[] distanceLoja;
    double geofenceSize = 200;
    float zoom = 15;
    Loja lojaAux;
    Produto produtoAux;
    LatLng markerLocation;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        permissionsAsk();

        DatabaseReference lojaReference = reference.child("loja");
```

```
        lojaReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                for (DataSnapshot objSnapshot : snapshot.getChildren()) {
                    lojaAux = new Loja();
                    lojaAux.setNome(String.valueOf(objSnapshot.child("nome").getValue()));
                    lojaAux.setLatitude(Double.parseDouble(String.valueOf(objSnapshot
                    .child("latitude").getValue())));
                    lojaAux.setLongitude(Double.parseDouble(String.valueOf(objSnapshot
                    .child("longitude").getValue())));
                    lojaAux.setID(String.valueOf(objSnapshot.getValue()));

                    for (DataSnapshot produtoSnapshot : objSnapshot
                    .child("produto").getChildren()) {
                    produtoAux = new Produto();
                    produtoAux.setModelo(String.valueOf(produtoSnapshot.child("modelo")
                    .getValue()));
                    produtoAux.setMarca(String.valueOf(produtoSnapshot.child("marca")
                    .getValue()));
                    produtoAux.setPreco(Double.parseDouble(String.valueOf(produtoSnapshot
                    .child("preco").getValue())));
                    produtoAux.setEspCaracter(String.valueOf(produtoSnapshot
                    .child("espCaracter").getValue()));
                    produtoAux.setImagemRef(String.valueOf(produtoSnapshot
                    .child("imagemRef").getValue()));
                     lojaAux.addProduto(produtoAux);
                  }

                   loja.add(lojaAux);
                }
                countLoja = (int) snapshot.getChildrenCount();
                distanceLoja = new int[countLoja];
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });

        client = LocationServices.getFusedLocationProviderClient(this);

        binding = ActivityMapsBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());

        SupportMapFragment mapFragment = (SupportMapFragment)
    getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
```

```
        seekBar = findViewById(R.id.seekBar);
        textSize = findViewById(R.id.textSize);
        buttonProdutos = findViewById(R.id.buttonProdutos);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_procurar_lojas, menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item) {
        boolean localizado;
        localizado = procurarLojas(item);
        if (localizado == false)
            return super.onOptionsItemSelected(item);
        else {
            if (item.getItemId() != R.id.itemProcurar && item.getItemId() != R.id.itemProdutos) {
                Intent intent = new Intent(getApplicationContext(), ProdutosLojaActivity.class);
                intent.putExtra("lojaList", (Serializable) loja);
                intent.putExtra("buscador", (Serializable) buscador);
                intent.putExtra("id", item.getItemId());
                startActivity(intent);
            }
        }
        return super.onOptionsItemSelected(item);
    }

    public boolean procurarLojas(MenuItem item) {
        float[] results = new float[1];
        Buscador auxBuscador;
        if (!markerLoja.isEmpty()) {
            for (int i = 0; i < markerLoja.size(); i++) {
                markerLoja.get(i).remove();
            }
            markerLoja.clear();
        }
        if (!buscador.isEmpty())
            buscador.clear();
        for (int i = 0; i < countLoja; i++) {
            Location.distanceBetween(markerLocation.latitude, markerLocation.longitude,
                    loja.get(i).getLatitude(), loja.get(i).getLongitude(), results);
            distanceLoja[i] = (int) results[0];
            if (results[0] <= geofenceSize) {
                if (item.getItemId() == R.id.itemProcurar) {
                    markerLoja.add(mMap.addMarker(new MarkerOptions().position(new
LatLng(loja.get(i).getLatitude(), loja.get(i).getLongitude()))
```

```
.title(loja.get(i).getNome()).icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorF
actory.HUE_YELLOW))
                        .snippet((int) results[0] + " metros de distância")));
            }
            auxBuscador = new Buscador();
            auxBuscador.setDistanciaLoja(distanceLoja[i]);
            auxBuscador.setNomeLoja(loja.get(i).getNome());
            auxBuscador.setPosicaoLoja(i);
            buscador.add(auxBuscador);
         }
      }
      if (buscador.isEmpty() && item.getItemId() != R.id.itemProdutos) {
         Toast.makeText(this, "Nenhuma loja encontrada", Toast.LENGTH_SHORT).show();
         return false;
      } else
         return true;
   }

   @Override
   protected void onResume() {
      super.onResume();

      //Condição exigida para ser possível utilizar o metodo getLastLocation
      if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
            ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
         return;
      }
      client.getLastLocation()
            .addOnSuccessListener(new OnSuccessListener<Location>() {
               @Override
               public void onSuccess(Location location) {
                  if (location != null)
                     Log.i("Location", "Location vazio");

                  LatLng myLocation = new LatLng(location.getLatitude(),
location.getLongitude());
                  home = myLocation;

                  if (firstTime) {
                     marker = mMap.addMarker(new
MarkerOptions().position(myLocation).title("Minha Localização"));

mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(myLocation, zoom));
                     circle = mMap.addCircle(new CircleOptions().center(myLocation)
                           .fillColor(Color.argb(20, 0, 255, 255))
```

```
                        .strokeWidth(8).radius(geofenceSize).visible(true));
                    markerLocation = home;
                    firstTime = false;
                }
            }
        }).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {

    }
});

    LocationRequest locationRequest = LocationRequest.create();
    locationRequest.setInterval(15 * 1000); //Intervalo de busca de atualização em
milisegundos
    locationRequest.setFastestInterval(5 * 1000); //Intervalo de busca em caso de
localizações vindas de outros apps

locationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURA
CY); //Economia maior de bateria com precisão de 100 metros

    //Verifica se as localizações recebidas estão corretas
    LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder()
        .addLocationRequest(locationRequest);

    SettingsClient settingsClient = LocationServices.getSettingsClient(this);
    settingsClient.checkLocationSettings(builder.build())
        .addOnSuccessListener(new OnSuccessListener<LocationSettingsResponse>() {
            @Override
            public void onSuccess(LocationSettingsResponse locationSettingsResponse) {
            }
        }).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        if (e instanceof ResolvableApiException) {
            try {
                ResolvableApiException resolvable = (ResolvableApiException) e;
                resolvable.startResolutionForResult(MapsActivity.this, 10);
            } catch (IntentSender.SendIntentException sendIntentException) {
            }
        }
    }
});

    LocationCallback locationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(@NonNull LocationResult locationResult) {
            super.onLocationResult(locationResult);
            if (locationResult == null) {
                Log.i("LocationResult", "locationResult vazio");
```

```
                return;
            }

            for (Location location : locationResult.getLocations()) {
                if (!Geocoder.isPresent()) {
                    return;
                }
            }
        }
    };
    client.requestLocationUpdates(locationRequest, locationCallback, null);
}

@Override
public void onMapReady(GoogleMap googleMap) {

    try {
        boolean success = googleMap.setMapStyle(
                MapStyleOptions.loadRawResourceStyle(
                        this, R.raw.style_json));

        if (!success) {
            Log.e("Estilo", "Style parsing failed.");
        }
    } catch (Resources.NotFoundException e) {
        Log.e("Estilo", "Can't find style. Error: ", e);
    }

    mMap = googleMap;

    mMap.setMinZoomPreference(10.0f);
    mMap.setMaxZoomPreference(20.0f);

    if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
            ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
        return;
    }
    mMap.setMyLocationEnabled(true);
    mMap.getUiSettings().setMyLocationButtonEnabled(true);
    mMap.getUiSettings().setZoomControlsEnabled(true);

    seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
        int minRadius;
        float zoomOut;

        @Override
```

```
            public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
                minRadius = 200 + i;
                zoomOut = zoom - (float) i / 1500;
                circle.setRadius(minRadius);
                if (zoomOut > 12)
                    mMap.animateCamera(CameraUpdateFactory.zoomTo(zoomOut));
                geofenceSize = minRadius;
                textSize.setText(minRadius + " m");
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {

            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {

            }
        });

        mMap.setOnMapLongClickListener(new GoogleMap.OnMapLongClickListener() {
            @Override
            public void onMapLongClick(@NonNull LatLng latLng) {
                geofenceMarker(latLng);
                mMap.animateCamera(CameraUpdateFactory.newLatLng(latLng));
                markerLocation = latLng;
            }
        });

        mMap.setOnMyLocationButtonClickListener(new
GoogleMap.OnMyLocationButtonClickListener() {
            @Override
            public boolean onMyLocationButtonClick() {
                geofenceMarker(home);
                markerLocation = home;
                return false;
            }
        });

        mMap.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener() {
            @Override
            public boolean onMarkerClick(@NonNull Marker markerClick) {
                if (!markerClick.getTitle().equals(marker.getTitle())) {
                    buttonProdutos.setVisibility(View.VISIBLE);
                } else {
                    buttonProdutos.setVisibility(View.INVISIBLE);
                }
                for (int i = 0; i < countLoja; i++) {
                    if (markerClick.getTitle().equals(loja.get(i).getNome())) {
```

```
                positionLoja = i;
            }
        }
        return false;
    }
});

buttonProdutos.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(getApplicationContext(), ProdutosLojaActivity.class);
        intent.putExtra("loja", loja.get(positionLoja));
        intent.putExtra("distancia", distanceLoja[positionLoja]);
        intent.putExtra("id", R.id.itemProcurar);
        startActivity(intent);
    }
});
}

public void permissionsAsk() {
    //Solicitar permissão de acesso a localização
    if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
        ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this, permissions, 1);
        return;
    }
}

public void geofenceMarker(LatLng latLng) {
    circle.remove();
    marker.remove();
    if (latLng == home)
        marker = mMap.addMarker(new MarkerOptions().position(latLng).title("Minha
Localização"));
    else
        marker = mMap.addMarker(new MarkerOptions().position(latLng).title("Marcador
Personalizado"));
    circle = mMap.addCircle(new CircleOptions().center(latLng)
        .fillColor(Color.argb(20, 0, 255, 255))
        .strokeWidth(8).radius(geofenceSize).visible(true));
}

//Alerta em caso de negar a permissão de acesso a localização
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
```

```
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);

        for (int resultado : grantResults) {
            if (resultado == PackageManager.PERMISSION_DENIED) {
                AlertDialog.Builder dialog = new AlertDialog.Builder(this);
                dialog.setTitle("Permissões negadas");
                dialog.setMessage("Para utilizar este aplicativo, é necessário aceitar as
permissões");
                dialog.setCancelable(false);

                dialog.setPositiveButton("CONFIRMAR", new DialogInterface.OnClickListener()
{
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {
                        finish();
                    }
                });
                dialog.create();
                dialog.show();
            }
        }
    }
}
```

APÊNDICE B – Código ProdutosLojasActivity.java

```
package com.tcc.geofencing.activity;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.DividerItemDecoration;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.TextView;
import com.tcc.geofencing.Buscador;
import com.tcc.geofencing.Loja;
import com.tcc.geofencing.Produto;
import com.tcc.geofencing.R;
import com.tcc.geofencing.adapter.Adapter;
import java.util.ArrayList;
import java.util.List;

public class ProdutosLojaActivity extends AppCompatActivity {

    private Loja loja;
    private List<Loja> lojaList;
    private List<Buscador> buscador;
    private int distancia;
    private RecyclerView recyclerView;
    private Adapter adapter;
    private List<Produto> produtoList = new ArrayList<Produto>();
    private TextView textFind;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_produtos_loja);

        recyclerView = findViewById(R.id.recyclerView);
        textFind = findViewById(R.id.textFind);

        Bundle dados = getIntent().getExtras();

        int id = dados.getInt("id");
        switch (id) {
            case R.id.itemProcurar:
                loja = (Loja) dados.getSerializable("loja");
                distancia = dados.getInt("distancia");
```

```
            adapter = new Adapter(loja.getProduto(), loja.getNome(), distancia);
            break;
        case R.id.itemHD:
            lojaList = (List<Loja>) dados.getSerializable("lojaList");
            buscador = (List<Buscador>) dados.getSerializable("buscador");
            carregarProdutos("HD Externo");
            if(produtoList.isEmpty())
                textFind.setVisibility(View.VISIBLE);
            adapter = new Adapter(produtoList, buscador);
            break;
        case R.id.itemRAM:
            lojaList = (List<Loja>) dados.getSerializable("lojaList");
            buscador = (List<Buscador>) dados.getSerializable("buscador");
            carregarProdutos("Memória RAM");
            if(produtoList.isEmpty())
                textFind.setVisibility(View.VISIBLE);
            adapter = new Adapter(produtoList, buscador);
            break;
        case R.id.itemMouse:
            lojaList = (List<Loja>) dados.getSerializable("lojaList");
            buscador = (List<Buscador>) dados.getSerializable("buscador");
            carregarProdutos("Mouse");
            if(produtoList.isEmpty())
                textFind.setVisibility(View.VISIBLE);
            adapter = new Adapter(produtoList, buscador);
            break;
        case R.id.itemFone:
            lojaList = (List<Loja>) dados.getSerializable("lojaList");
            buscador = (List<Buscador>) dados.getSerializable("buscador");
            carregarProdutos("Fone de Ouvido");
            if(produtoList.isEmpty())
                textFind.setVisibility(View.VISIBLE);
            adapter = new Adapter(produtoList, buscador);
            break;
        case R.id.itemProcessador:
            lojaList = (List<Loja>) dados.getSerializable("lojaList");
            buscador = (List<Buscador>) dados.getSerializable("buscador");
            carregarProdutos("Processador");
            if(produtoList.isEmpty())
                textFind.setVisibility(View.VISIBLE);
            adapter = new Adapter(produtoList, buscador);
            break;
    }

    RecyclerView.LayoutManager layoutManager =
    new LinearLayoutManager(getApplicationContext());
    recyclerView.setLayoutManager(layoutManager);
    recyclerView.setHasFixedSize(true);
    recyclerView.addItemDecoration(new DividerItemDecoration
    (this, LinearLayout.VERTICAL));
```

```
    recyclerView.setAdapter(adapter);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_ordenar_produtos, menu);
    if (buscador == null || buscador.isEmpty()) {
        invalidateOptionsMenu();
        MenuItem item = menu.findItem(R.id.itemOrdenar);
        item.setVisible(false);
    }
    return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    switch (item.getItemId()) {
        case R.id.itemDistancia:
            ordenarDistancia();
            adapter.notifyDataSetChanged();
            break;
        case R.id.itemMenorPreco:
            ordenarMenorPreco();
            adapter.notifyDataSetChanged();
            break;
        case R.id.itemMaiorPreco:
            ordenarMaiorPreco();
            adapter.notifyDataSetChanged();
            break;
    }
    return super.onOptionsItemSelected(item);
}

public void ordenarDistancia() {
    boolean troca = true;
    Produto aux;
    double distancia1;
    double distancia2;
    int j;
    int k;
    while (troca) {
        troca = false;
        for (int i = 0; i < produtoList.size() - 1; i++) {
            j = produtoList.get(i).getIdBuscador();
            k = produtoList.get(i + 1).getIdBuscador();
            distancia1 = buscador.get(j).getDistanciaLoja();
            distancia2 = buscador.get(k).getDistanciaLoja();
            if (distancia1 > distancia2) {
                aux = produtoList.get(i);
                produtoList.set(i, produtoList.get(i + 1));
```

```
                produtoList.set(i + 1, aux);
                troca = true;
            }
        }
    }
}

public void ordenarMenorPreco() {
    boolean troca = true;
    Produto aux;
    while (troca) {
        troca = false;
        for (int i = 0; i < produtoList.size() - 1; i++) {
            double preco1 = produtoList.get(i).getPreco();
            double preco2 = produtoList.get(i + 1).getPreco();
            if (preco1 > preco2) {
                aux = produtoList.get(i);
                produtoList.set(i, produtoList.get(i + 1));
                produtoList.set(i + 1, aux);
                troca = true;
            }
        }
    }
}

public void ordenarMaiorPreco() {
    boolean troca = true;
    Produto aux;
    while (troca) {
        troca = false;
        for (int i = 0; i < produtoList.size() - 1; i++) {
            double preco1 = produtoList.get(i).getPreco();
            double preco2 = produtoList.get(i + 1).getPreco();
            if (preco1 < preco2) {
                aux = produtoList.get(i);
                produtoList.set(i, produtoList.get(i + 1));
                produtoList.set(i + 1, aux);
                troca = true;
            }
        }
    }
}

public void carregarProdutos(String modeloProduto) {
    int j;
    for (int i = 0; i < buscador.size(); i++) {
        j = buscador.get(i).getPosicaoLoja();
        for (Produto produto : lojaList.get(j).getProduto()) {
            if (produto.getModelo().equals(modeloProduto)) {
                produto.setIdBuscador(i);
```

```
                produtoList.add(produto);
            }
        }
    }
}
```

APÊNDICE C – Código Adapter.java

```
package com.tcc.geofencing.adapter;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.firebase.storage.FileDownloadTask;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import com.tcc.geofencing.Buscador;
import com.tcc.geofencing.Produto;
import com.tcc.geofencing.R;
import java.io.File;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.List;

public class Adapter extends RecyclerView.Adapter<Adapter.MyViewHolder> {

    private List<Produto> listaProduto;
    private String nomeLoja;
    private int distancia;
    private StorageReference imagensProdutos;
    private List<Buscador> buscador;
    private DecimalFormat decimalFormat = new DecimalFormat("####.00");

    public Adapter(List<Produto> listaProduto, String nomeLoja, int distancia) {
        this.listaProduto = listaProduto;
        this.nomeLoja = nomeLoja;
        this.distancia = distancia;
    }

    public Adapter(List<Produto> listaProduto, List<Buscador> buscador) {
        this.listaProduto = listaProduto;
        this.buscador = buscador;
    }

    @NonNull
    @Override
```

```
    public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType)
{
        View itemLista = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.adapter_lista, parent, false);
        return new MyViewHolder(itemLista);
    }

    @Override
    public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
        Produto produto = listaProduto.get(position);

        carregarImagem(produto.getImagemRef(), holder);

        if(buscador == null) {
            holder.nomeLoja.setText(nomeLoja);
            holder.distancia.setText("<" + distancia + " m");
        } else {
            int aux = listaProduto.get(position).getIdBuscador();
            holder.nomeLoja.setText(buscador.get(aux).getNomeLoja());
            holder.distancia.setText("<" + buscador.get(aux).getDistanciaLoja() + " m");
        }
        holder.modeloMarca.setText(produto.getModelo() + " " + produto.getMarca());
        holder.preco.setText("R$ " + decimalFormat.format(produto.getPreco())
        .replace("." , ","));
        holder.espec.setText(produto.getEspCaracter());
    }

    @Override
    public int getItemCount() {
        return listaProduto.size();
    }

    public class MyViewHolder extends RecyclerView.ViewHolder{
        TextView modeloMarca, espec, nomeLoja, distancia, preco;
        ImageView foto;

        public MyViewHolder(@NonNull View itemView) {
            super(itemView);
            modeloMarca = itemView.findViewById(R.id.textModeloMarca);
            espec = itemView.findViewById(R.id.textEspec);
            nomeLoja = itemView.findViewById(R.id.textNomeLoja);
            distancia = itemView.findViewById(R.id.textDistancia);
            preco = itemView.findViewById(R.id.textPreco);
            foto = itemView.findViewById(R.id.imageProduto);
        }
    }

    public void carregarImagem(String imagemRef, MyViewHolder holder) {
        imagensProdutos = FirebaseStorage.getInstance().getReference()
        .child("images/" + imagemRef);
```

```
        try {
            final File localFile = File.createTempFile("tempfile", ".jpg");
            imagensProdutos.getFile(localFile)
                    .addOnSuccessListener(new
OnSuccessListener<FileDownloadTask.TaskSnapshot>() {
                @Override
                public void onSuccess(FileDownloadTask.TaskSnapshot taskSnapshot) {
                    Bitmap bitmap = BitmapFactory.decodeFile(localFile.getAbsolutePath());
                    holder.foto.setImageBitmap(bitmap);
                }
            }).addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Log.e("Image Failure", e.toString());
                }
            });
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

APÊNDICE D – Código Loja.java

```
package com.tcc.geofencing;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class Loja implements Serializable {

   private String nome;
   private double latitude;
   private double longitude;
   private List<Produto> produto = new ArrayList<Produto>();
   private String ID;

   public String getNome() {
      return nome;
   }

   public void setNome(String nome) {
      this.nome = nome;
   }

   public double getLatitude() {
      return latitude;
   }

   public void setLatitude(double latitude) {
      this.latitude = latitude;
   }

   public double getLongitude() {
      return longitude;
   }

   public void setLongitude(double longitude) {
      this.longitude = longitude;
   }

   public List<Produto> getProduto() {
      return produto;
   }

   public void setProduto(List<Produto> produto) {
      this.produto = produto;
   }

   public String getID() {
```

```
        return ID;
    }

    public void setID(String ID) {
        this.ID = ID;
    }

    public void addProduto (Produto obj) {
        produto.add(obj);
    }
}
```

APÊNDICE E – Código Produto.java

```
package com.tcc.geofencing;

import java.io.Serializable;

public class Produto implements Serializable {

   private String modelo;
   private String marca;
   private Double preco;
   private String espCaracter;
   private String imagemRef;
   private int idBuscador;

   public String getModelo() {
      return modelo;
   }

   public void setModelo(String modelo) {
      this.modelo = modelo;
   }

   public String getMarca() {
      return marca;
   }

   public void setMarca(String marca) {
      this.marca = marca;
   }

   public Double getPreco() {
      return preco;
   }

   public void setPreco(Double preco) {
      this.preco = preco;
   }

   public String getEspCaracter() {
      return espCaracter;
   }

   public void setEspCaracter(String espCaracter) {
      this.espCaracter = espCaracter;
   }

   public String getImagemRef() {
      return imagemRef;
```

```
    }

    public void setImagemRef(String imagemRef) {
        this.imagemRef = imagemRef;
    }

    public int getIdBuscador() {
        return idBuscador;
    }

    public void setIdBuscador(int idBuscador) {
        this.idBuscador = idBuscador;
    }
}
```

APÊNDICE F – Código Buscador.java

```
package com.tcc.geofencing;

import java.io.Serializable;

public class Buscador implements Serializable {

   private String nomeLoja;
   private int distanciaLoja;
   private int posicaoLoja;

   public String getNomeLoja() {
      return nomeLoja;
   }

   public void setNomeLoja(String nomeLoja) {
      this.nomeLoja = nomeLoja;
   }

   public int getDistanciaLoja() {
      return distanciaLoja;
   }

   public void setDistanciaLoja(int distanciaLoja) {
      this.distanciaLoja = distanciaLoja;
   }

   public int getPosicaoLoja() {
      return posicaoLoja;
   }

   public void setPosicaoLoja(int posicaoLoja) {
      this.posicaoLoja = posicaoLoja;
   }
}
```

APÊNDICE G – Códigos da Interface

Pasta layout – activity_maps.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:map="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".activity.MapsActivity">

   <fragment
      android:id="@+id/map"
      android:name="com.google.android.gms.maps.SupportMapFragment"
      android:layout_width="0dp"
      android:layout_height="0dp"
      map:layout_constraintBottom_toTopOf="@+id/seekBar"
      map:layout_constraintEnd_toEndOf="parent"
      map:layout_constraintHorizontal_bias="1.0"
      map:layout_constraintStart_toStartOf="parent"
      map:layout_constraintTop_toTopOf="parent"
      map:layout_constraintVertical_bias="1.0"
      tools:context=".activity.MapsActivity" />

   <SeekBar
      android:id="@+id/seekBar"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:background="@color/purple_700"
      android:max="4800"
      android:padding="16dp"
      android:paddingStart="32dp"
      android:progress="0"
      android:progressTint="@color/white"
      android:thumbTint="@color/white"
      map:layout_constraintBottom_toBottomOf="parent"
      map:layout_constraintEnd_toStartOf="@+id/textSize"
      map:layout_constraintStart_toStartOf="parent" />

   <TextView
      android:id="@+id/textSize"
      android:layout_width="100dp"
      android:layout_height="0dp"
      android:background="@color/purple_700"
      android:gravity="center"
      android:text="200 m"
```

```
        android:textColor="@color/white"
        android:textSize="20sp"
        android:textStyle="bold"
        map:layout_constraintBottom_toBottomOf="parent"
        map:layout_constraintEnd_toEndOf="parent"
        map:layout_constraintTop_toTopOf="@+id/seekBar" />

    <Button
        android:id="@+id/buttonProdutos"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="6dp"
        android:text="Ver produtos"
        android:visibility="invisible"
        map:layout_constraintStart_toStartOf="parent"
        map:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Pasta layout – activity_produtos_loja.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activity.ProdutosLojaActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textFind"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Nenhum produto encontrado"
        android:textSize="20sp"
        android:textStyle="bold"
        android:visibility="invisible"
        app:layout_constraintBottom_toBottomOf="@+id/recyclerView"
```

```
        app:layout_constraintEnd_toEndOf="@+id/recyclerView"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Pasta layout – adapter_lista.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">

        <ImageView
            android:id="@+id/imageProduto"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_weight="0"
            android:scaleType="fitCenter"
            tools:srcCompat="@tools:sample/avatars" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="0"
            android:orientation="vertical">

            <TextView
                android:id="@+id/textModeloMarca"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="8dp"
                android:text="Modelo e Marca"
                android:textSize="18sp"
                android:textStyle="bold" />

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_weight="0"
                android:orientation="horizontal">
```

```
        <TextView
            android:id="@+id/textEspec"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"
            android:layout_weight="1"
            android:text="Especificação"
            android:textSize="16sp" />

        <TextView
            android:id="@+id/textPreco"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:layout_weight="0"
            android:text="Preço"
            android:textSize="16sp"
            android:textStyle="bold" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_weight="0"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textNomeLoja"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"
            android:layout_weight="1"
            android:text="Nome da Loja"
            android:textSize="12sp" />

        <TextView
            android:id="@+id/textDistancia"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:layout_weight="0"
            android:text="Distância"
            android:textSize="10sp" />
    </LinearLayout>

</LinearLayout>
```

```
   </LinearLayout>
</LinearLayout>
```

Pasta menu – menu_procurar_lojas.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto">

   <item
      android:id="@+id/itemProcurar"
      android:title="Procurar Lojas"
      app:showAsAction="always"
      />

   <item
      android:id="@+id/itemProdutos"
      android:title="Procurar Produtos"
      android:icon="@drawable/ic_buscar_24"
      app:showAsAction="always">

      <menu>
         <item
            android:id="@+id/itemHD"
            android:title="HD Externo"
            android:orderInCategory="100"
            />

         <item
            android:id="@+id/itemRAM"
            android:title="Memória RAM"
            android:orderInCategory="200"
            />

         <item
            android:id="@+id/itemMouse"
            android:title="Mouse"
            android:orderInCategory="300"
            />

         <item
            android:id="@+id/itemFone"
            android:title="Fone de Ouvido"
            android:orderInCategory="400"
            />

         <item
            android:id="@+id/itemProcessador"
```

```
            android:title="Processador"
            android:orderInCategory="500"
            />

      </menu>
   </item>

</menu>
```

Pasta menu – menu_ordenar_produtos.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto">

   <item
      android:id="@+id/itemOrdenar"
      android:title="Ordenar"
      app:showAsAction="always|withText">

      <menu>

         <item
            android:id="@+id/itemDistancia"
            android:orderInCategory="100"
            android:title="Distância" />

         <item
            android:id="@+id/itemMenorPreco"
            android:orderInCategory="200"
            android:title="Menor Preço" />

         <item
            android:id="@+id/itemMaiorPreco"
            android:orderInCategory="300"
            android:title="Maior Preço" />
      </menu>

   </item>

</menu>
```

Pasta raw – style_json.json

```
[
  {
    "featureType": "poi",
    "elementType": "all",
    "stylers": [
      {
        "visibility": "off"
      }
    ]
  },
  {
    "featureType": "transit",
    "elementType": "labels.icon",
    "stylers": [
      {
        "visibility": "off"
      }
    ]
  }
]
```

APÊNDICE H – Código AndroidManifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tcc.geofencing">

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Geofencing"
        android:screenOrientation="portrait">
        <activity
            android:name=".activity.ProdutosLojaActivity"
            android:exported="false" />

        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <activity
            android:name=".activity.MapsActivity"
            android:exported="true"
            android:label="@string/title_activity_maps">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

APÊNDICE I – Código build.gradle (Project: Geofencing)

```
// Top-level build file where you can add configuration options common to all sub-
projects/modules.
buildscript {
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:7.0.3'
        classpath 'com.google.gms:google-services:4.3.10'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

APÊNDICE J – Código build.gradle (Module: Geofencing.app)

```
plugins {
    id 'com.android.application'
}

android {
    compileSdk 30

    defaultConfig {
        applicationId "com.tcc.geofencing"
        minSdk 21
        targetSdk 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-
rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    buildFeatures {
        viewBinding true
    }
}

apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'

dependencies {

    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'com.google.android.gms:play-services-maps:17.0.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.0'
    implementation 'com.google.firebase:firebase-storage:19.1.1'
    implementation 'com.google.firebase:firebase-database:19.2.1'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
```

```
    implementation 'com.google.android.gms:play-services-maps:17.0.1'
    implementation 'com.google.android.gms:play-services-location:18.0.0'

    implementation platform('com.google.firebase:firebase-bom:28.4.0')
    implementation 'com.google.firebase:firebase-analytics'
}
```