INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Personal RoadBook: uma aplicação para o registo de experiências

Mariana Cândido Vaz

Mestrado em Engenharia de Telecomunicações e Informática,

Orientador:
Professor Doutor Luís Henrique Ramilo Mota, Professor Auxiliar,
Iscte – Instituto Universitário de Lisboa

Novembro, 2021

TECNOLOGIAS
E ARQUITETURA

Departamento de Ciências e Tecnologias da Informação

Personal RoadBook: uma aplicação para o registo de experiências

Mariana Cândido Vaz

Mestrado em Engenharia de Telecomunicações e Informática,

Orientador:
Professor Doutor Luís Henrique Ramilo Mota, Professor Auxiliar,
Iscte – Instituto Universitário de Lisboa

Novembro, 2021

Dedico esta dissertação aos meus pais que sempre me apoiaram e proporcionaram o sucesso do meu percurso académico.

Agradecimento

Gostaria de começar por agradecer aos meus pais. Proporcionaram todas as condições para que o meu percurso académico fosse possível. Toda a paciência, apoio e incentivo foram importantes para finalizar esta etapa.

Um especial obrigado a todos os meus familiares e amigos que me acompanharam ao longo destes anos, todos os momentos que me proporcionaram foram importantes. Aos amigos universitários agradeço toda a troca de conhecimentos, toda a partilha e todo o apoio.

Agradeço também ao Iscte - Instituto Universitário de Lisboa por todas as aulas, todas as condições e todo o conhecimento que tornou possível a realização deste trabalho.

Por último, mas não menos importante, gostaria de agradecer ao meu orientador Prof. Dr. Luís Mota por toda disponibilidade, compreensão e ajuda que levou ao sucesso desta dissertação.

A todos o meu sincero obrigado!

Resumo

O crescimento da *Internet* tem alterado o conceito de comunicação na sociedade, tornando-a mais rápida e alargada a qualquer lugar no mundo. Proporciona disponibilidade abundante de dados e tem cada vez mais utilizadores. As redes sociais, por seu lado, proporcionaram o aparecimento de comunidades virtuais entre indivíduos que partilham interesses comuns. A partilha de experiências entre indivíduos conhecidos tem sido valorizada desde sempre, e é caracterizada pela confiança nas informações dadas. Estes fatores têm contribuído positivamente para uma nova forma de abordar o turismo, aumentando o suporte de informações turísticas disponíveis *online* que ajudam os turistas na escolha de destinos para férias e planeamento de rotas e experiências. Neste contexto foi desenvolvida esta dissertação, onde foi proposto e implementado um Sistema de Informação Geográfica no formato de rede social.

A implementação do projeto baseia-se numa aplicação *Android* onde é possível a partilha de dados com componente geográfica entre amigos. Uma aplicação onde é apresentado um mapa que permite a criação e consulta de marcadores. Cada marcador é composto por comentários e fotografias partilhadas pelos utilizadores da aplicação, e a sua visualização é gerida através da privacidade atribuída ao marcador.

Após a implementação foi elaborado um questionário para testar a aplicação e reunir resultados que foram analisados.

Palavras-Chave: Sistema de Informação Geográfica; Aplicação *Android*; Rede Social; *Firebase*.

Abstract

The growth of the Internet has changed the concept of communication in present days, making it faster and more widespread anywhere in the world. It provides abundant availability of data and has gained more and more users. Social networks have created the conditions for emergence of virtual communities between persons who share common interests or personal acquaintance. The sharing of experiences between related persons has always been valued, and is characterized by the reliability of the information given. These factors have contributed positively to a new attitude towards tourism, by increasing the support of touristic information available online that helps tourists choosing holiday destinations. In this context, this dissertation was developed, where a Geographic Information System in a social network format was designed and implemented.

The project implementation consists of an Android application where it is possible to share geographic data related to tourism and leisure between friends. An application where a map is presented that allows the creation and query of markers. Each marker is associated with comments and photos shared by the users of the application, and their visibility depends on the privacy assigned to the marker.

A questionnaire was designed to evaluate the application and gather results that were analyzed.

Keywords: Geographic Information System; Android application; Social network; Firebase.

Índice

Lista de Figuras

Glossário

API - *Application Programming Interface*
BaaS - *Backend as a Service*
CSV - *Comma-Separated Values*
CAD - *Computer-Aided Design*
CRUD - *Create, Read, Update and Delete*
DSR - *Design Science Research*
DBMS - *Database Management System*
ESRI - *Environmental Systems Research Institute*
GML - *Geography Markup Language*
GPS - *Global Positioning System*
IDE - *Integrated Development Environment*
IL - Inteligência de Localização
JSON - *JavaScript Object Notation*
JSONB - *JavaScript Object Notation Binary*
KML - *Keyhole Markup Language*
LBS - *Location-based Service*
NoSQL - *Not Only Structured Query Language*
OGC - *Open Geospatial Consortium*
Renderizar - *Render*
RSS - *Really Simple Syndication*
RSSI - *Received Signal Strength Indicator*
SDK - *Software development kit*
SIG - Sistema de Informação Geográfica
SQL - *Structured Query Language*
UI - *User Interface*
WMS - *Web Mapping Service*
WKT - *Well-known text*
WPS - *Wi-Fi Positioning System*
XML - *Extensible Markup Language*

CAPíTULO 1

Introdução

1.1. Motivação

Os meios de comunicação têm contribuído desde há vários anos para a partilha de informação entre indivíduos. O seu desenvolvimento tem sido notável e tem contribuído de forma positiva para todas as áreas da sociedade, e em particular para o turismo. Os jornais e revistas eram uma das grandes referências para a escolha de destinos de férias pelos turistas. O aparecimento da *Internet* 'aproximou pessoas e locais' [**1**] e teve uma adesão imensa de utilizadores. Ao mesmo tempo, houve uma enchente de informação com as mais variadas fontes. A título de exemplo, a partilha de informação por parte de celebridades continua a contribuir para o turismo ao gerar o desejo por parte dos seus seguidores de conhecerem os lugares por eles frequentados. Na sociedade em que estamos inseridos sempre se valorizou o passa a palavra, 'caracterizado pela veracidade e confiança das informações dadas' [**1**], e tendo em conta que 'a credibilidade de um determinado assunto aumenta quanto maior for a confiança na pessoa que fornece a informação' [**1**], a partilha da informação por parte de pessoas que nos são conhecidas contribui para esse fator [**1**].

Tendo em conta a situação pandémica que vivemos, muitas pessoas procuram o turismo como um escape. 'Experiências proporcionadas pelo turismo podem ser aliadas no cuidado com a saúde mental e ajudar a lidar com quadros provenientes da pandemia, como ansiedade e exaustão.' afirma Nathalia Molina em [**2**].

De modo a facilitar a navegação pela informação, a implementação desta aplicação baseia-se na criação de informação por parte de familiares ou amigos. É adotada a sua apresentação de forma geolocalizada, sobre um mapa, para facilitar a sua perceção.

Pretende-se testar a criação de um serviço descentralizado, baseado na partilha de experiências pessoais sobre viagens ou lugares. Um serviço que funciona como uma rede social de viagens, onde a organização e a fácil partilha de informação entre utilizadores conhecidos são o foco principal. Cada utilizador tem a possibilidade de criar o seu próprio repositório com os conteúdos que identificar relevantes. Estes, por sua vez, estarão disponíveis na sua rede de amigos. Os conteúdos são protegidos por uma política de acessos gerida pela autenticação de utilizadores e por um parâmetro de privacidade implementado que permite que o próprio utilizador tenha controlo sobre a informação partilhada com cada amigo.

Atualmente, o número de utilizadores de *smartphones* no mundo ultrapassa os mil milhões e há uma previsão de crescimento de centenas de milhões nos próximos anos [**3**].

A conveniência de usar um *smartphone* e a necessidade de partilha de informação foram o ponto de partida para a criação da aplicação Diário de Viagens (*Personal Roadbook*).

1.2. Enquadramento

Este projeto baseia-se numa aplicação onde é implementado um Sistema de Informação Geográfica (SIG), com acesso e manipulação distribuídos, para *smartphones*, que são dispositivos móveis de bolso e de fácil acesso. O sistema operativo *Android* para *smartphones*, apesar da concorrência elevada no mercado, continua a ser a opção mais popular [**4**]. Por isso foi escolhido para base do desenvolvimento, mas nada impede o desenvolvimento para outros SOs.

Um Sistema de Informação Geográfica é uma estrutura para recolha, gestão e análise de dados com uma componente de localização relevante. É um sistema que integra vários tipos de dados e ajuda o utilizador na interpretação de dados, a partir do momento em que analisa a geolocalização, localização geográfica de um indivíduo ou de um dispositivo, e organiza camadas de informação sobre mapas [**5**].

Dois mecanismos de geolocalização possíveis são:

- O *Global Positioning System* (GPS). O GPS é um sistema de navegação por satélite que fornece a um dispositivo recetor móvel a sua localização. A localização é determinada pelo recetor GPS através do cálculo das diferenças temporais dos vários sinais de rádio trocados com os satélites, esses sinais são enviados constantemente e neles está presente o momento de emissão do satélite [**6, 7**]. Este sistema está disponível em virtualmente todos os *smartphones*.

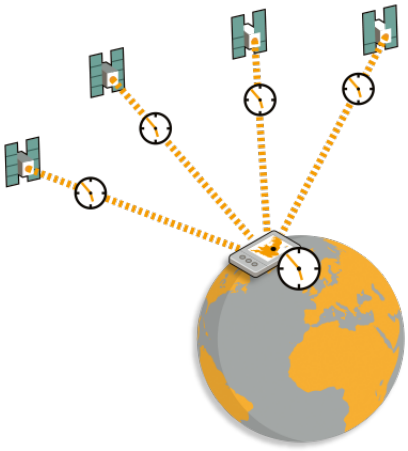


FIGURA 1. Representação do funcionamento do GPS.

- *Wi-Fi Positioning System* (WPS). Por vezes, em alternativa ou complemento ao GPS é utilizado o WPS que, sendo um sistema de geolocalização que usa características de *hotspots Wi-Fi* próximos e outros *hotspots* sem fios, apresenta-se vantajoso na determinação da localização em ambientes fechados. Uma das técnicas utilizadas baseia-se em RSSI (*Received Signal Strength Indicator* - Indicador da Intensidade do Sinal Recebido), onde as medições da intensidade do sinal

são guardadas e a posterior análise dessas medições determina a localização do dispositivo móvel [**8**]. A localização pode também, em alternativa, ser calculada a partir do sinal da rede móvel.

1.3. Questões de Investigação

Nesta secção são desenvolvidas questões que ajudam ao desenvolvimento do projeto:

- Como organizar dados geográficos de forma eficiente?
- Que impacto terá a interação entre utilizadores?
- Qual o impacto da apresentação de mapas com a informação disponibilizada pelos utilizadores?

1.4. Objetivos

Este projeto visa servir como um repositório de criação de dados de experiências e planos para locais específicos por onde o utilizador viajou, bem como apresentação e consulta dos mesmos, tornando uma futura experiência por parte de outro utilizador mais rica, sendo apresentado sob a forma de aplicação móvel.

A aplicação móvel pretende ser intuitiva, de fácil acesso e gratuita, estendendo-se a todas as faixas etárias que possuam *smartphones*. De facto, quantos mais utilizadores alcançar, mais proveitosa se torna a sua utilização, pois mais informações se encontrarão disponíveis. Caso exista muita informação, a navegação pode, porém, ser mais complicada e será um fator a ter em conta.

O desenvolvimento desta aplicação móvel requer a utilização de tecnologias para a implementação da geolocalização e o acesso à *internet*.

CAPíTULO 2

Revisão da Literatura

Esta secção descreve a pesquisa com base em artigos, tecnologias e aplicações relativas a projetos semelhantes ou relacionados ao proposto, de modo a expor e assimilar o conteúdo que existe no mercado relativamente ao tema.

2.1. Artigos

Em "A Web and Mobile GIS for Identifying Areas within the Radius Affected by Natural Disasters Based on OpenStreetMap Data"é proposto um SIG que identifica áreas afetadas por desastres naturais delineadas por um raio de impacto na Indonésia, disponibilizado em *interface Web* e aplicação móvel *Android*. Como camada de mapas são utilizados os dados do *OpenStreetMap* e para os armazenar é utilizada a base de dados espaciais *PostgreSQL*, na qual foi acrescentada a extensão *PostGIS*. Depois de processados na base de dados, os dados espaciais são apresentados como mapas nas plataformas através da biblioteca *Leaflet*. A aplicação móvel acede aos serviços da *web* através de uma *interface* com a base de dados *PostgreSQL* [**9**].

Em "Location Based Services and Integration of Google Maps in Android"é feito um estudo detalhado sobre componentes imprescindíveis para ser possível a implementação de *Location-based Service* (LBS) numa aplicação *Android*. É explicado também o uso e a implementação do *Google Maps*, bem como APIs para obter várias informações de localização com base no *Android*. LBS usa tecnologias como GPS, redes de comunicação e *Wi-Fi* para prestar vários serviços. O pacote *android.location*, contido no *Android*, fornece a API para determinar a geolocalização atual. O *Android* permite a integração do *Google Maps*, através do uso de uma biblioteca de serviços *Google Play*, providenciando funcionalidades como mostrar localizações no mapa ou mostrar diferentes rotas no mapa [**10**].

Em "Navegação indoor baseada na rede wifi como suporte a serviços baseados na localização: estudo de caso no campus da UL"é apresentada uma solução de um serviço baseado na localização em ambientes fechados, baseado na rede *Wi-Fi*, para o *campus* da Universidade de Lisboa. É criada uma aplicação móvel, desenvolvida para *Android*, desempenhando um serviço baseado na localização do utilizador. Para a disponibilização dos mapas para a *interface* da aplicação foi utilizado o *software ArcGIS Server* e a base de dados espacial foi criada através do *software PostGreSQL 9.2* com extensão *PostGIS 2.0*. O sistema é dividido em três componentes distintos, o componente do cliente, do servidor e da base de dados. O componente do cliente é constituído por uma aplicação móvel, e por uma base de dados local, disponibilizada através de uma biblioteca existente para o

Android, chamada *SQLite*, onde é guardada informação relativa à posição do utilizador. A informação espacial é carregada através do programa *PostGIS 2.0 Shapefile* e *DBF Loader* [**11**].

A análise destes três artigos contribuiu para a descoberta de tecnologias utilizadas e com suporte para dados geográficos. No terceiro trabalho foi ainda possível compreender uma estrutura para integração dos diferentes componentes essenciais a um serviço geolocalizado.

2.2. Tecnologias

De acordo com as necessidades atuais, todas as tecnologias têm sofrido alterações no seu funcionamento de modo a corresponderem às expectativas do utilizador. Para tornar as aplicações mais responsivas e menos dependentes da rede recorre-se cada vez mais ao uso *offline*, uma tendência que tem ganho grande popularidade. O uso *offline* baseia-se na apresentação de dados presentes na base de dados sem existir ligação à *Internet*. Para que tal seja possível é necessária uma cópia da base de dados no dispositivo que é sincronizada sempre que a ligação à *Internet* seja possível [**12**].

Sendo o armazenamento dos dispositivos móveis limitado, procuram-se bases de dados rápidas, seguras e leves, com baixo consumo de energia. Pretende-se que as suas dependências sejam mínimas e o seu código fácil, de modo a facilitar o seu uso e partilha com outros dispositivos móveis [**12**].

Uma vez que o projeto se baseia na representação de informação geográfica, a base de dados a usar necessita de estar adaptada a este tipo de dados. Surge então o termo de base de dados espacial ou Sistema de Informação Geográfica que se baseia num sistema de informação otimizado para armazenamento e consulta de dados espaciais, ou seja, que representa objetos geográficos e funcionalidades para os processar. A complexidade da mesma pode variar desde a representação de elementos mais simples como pontos e linhas até à representação de elementos mais complexos como objetos 3D. Este tipo de base de dados distingue-se das restantes por requerer uma funcionalidade adicional que proporciona maior eficiência no processamento de dados do tipo espacial. Esta funcionalidade é suportada por padrões criados pela *Open Geospatial Consortium* (OGC) a partir de 1997, de acordo com [**13**].

2.2.1. Bases de dados espaciais

Dois exemplos de bases de dados espaciais com uma complexidade simples são: *MongoDB* e *Couchbase Lite*, que se baseiam num sistema não relacional.

Como exemplos de bases de dados mais robustas, ou seja, que se baseiam numa estrutura relacional, são apresentadas as seguintes: *SQLite* e *PostgreSQL*.

MongoDB

MongoDB é uma base de dados não relacional, baseada em documentos JSON e com um modelo de desenvolvimento de código aberto. Oferece suporte para tratamento de dados espaciais [**14**]. Para guardar dados espaciais recorre-se a objetos *GeoJSON*, que

suportam dados do tipo Ponto, Sequência de Pontos, Polígono, MultiPointo, Conjunto de Sequência de Ponto, MultiPolígono e Coleção de Geometria, ou a pares de coordenadas com índice 2D. Sobre estes tipos de dados é disponibilizado o conjunto de operações denominado CRUD (*Create, Read, Update and Delete*), sendo as quatro operações básicas oferecidas pelas bases de dados: criar, ler, atualizar e apagar. É um sistema multiplataforma que por si só não abrange *Android*. Contudo, a sua extensão a um sistema *Android* torna-se possível através da ponte efetuada pelo serviço *MongoDB Mobile*, que fornece uma API compatível com *Android* [**15**].

Couchbase Lite

Couchbase Lite é uma base de dados não relacional, baseada em documentos JSON. Considera-se um sistema de código aberto e multiplataforma, que implementa características da base de dados *Couchbase*. Tem a capacidade de sincronização integrada e os dados apresentam-se disponíveis *offline* na base de dados local. A sua API é de fácil utilização, suportando diferentes tipos de dados: *numbers*, *strings* ou *arrays*. Sobre estes tipos de dados é disponibilizado o conjunto de operações CRUD. É considerada uma base de dados leve [**16**].

Esta base de dados necessita de uma extensão para interpretar dados espaciais. *GeoCouch* é uma extensão espacial para *Couchbase*, que torna possível a interpretação de dados espaciais como vetores, concedendo-lhe rapidez de processamento relativamente a este tipo de dados. Não tem capacidade para interpretar dados mais complexos como polígonos, para tal necessitaria de implementar uma biblioteca geométrica [**17**].

SQLite

SQLite é uma base de dados relacional. Implementa uma biblioteca de linguagem C, sendo considerado um sistema de código aberto e multiplataforma. Não necessita de implementar um servidor, pois gere autonomamente os processos de leitura e escrita através de cadeados. É permitido o acesso múltiplo a processos de leitura. Quando surge um processo de escrita o acesso ao ficheiro é bloqueado durante os segundos da atualização. As operações disponibilizadas são de acordo com as características de implementação SQL, de entre um vasto conjunto pode-se considerar: ordenar por (*order by*), agrupar por (*group by*), atualizar, apagar, inserir ou operações de agregação que assentam sobre tipos de dados simples como *numbers* ou *strings*. No *SQLite* é possível indexar dados multidimensionais como coordenadas geográficas, retângulos ou polígonos [**18**].

Esta base de dados necessita de uma extensão para interpretar dados espaciais. *SpatiaLite* é uma extensão espacial para *SQLite*, fornecendo funcionalidades de base de dados espacial vetorial, compatível com *Android*. Aumenta o suporte espacial existente do *SQLite*, sendo necessário para consultas espaciais avançadas e para suportar múltiplas projeções de mapas. Fornece a implementação de uma ferramenta SIG para pesquisa de dados e troca de dados geoespaciais [**19**].

PostgreSQL

PostgreSQL é uma base de dados relacional que usa linguagem SQL. Teve origem na Universidade da Califórnia em *Berkeley* no ano de 1986, tendo mais de 30 anos de desenvolvimento. Encontra-se como código aberto e disponibilizada gratuitamente. Permite o manuseamento de diferentes tipos de dados: primitivos (*Integer, Numeric, String, Boolean*), estruturados (Data/Hora, Matriz), documentos (JSON/JSONB, XML) e geometria (*Point, Line, Circle, Polygon*) [**20**].

PostGIS é uma extensão espacial para *PostgreSQL*, de código aberto, que adiciona novos tipos de dados (geográficos, geométricos, *raster* e outros), funções, operadores e melhorias de índice, como velocidade de acesso mais rápida, à base de dados referida. Exemplos destas adições são: 'renderização e importação de funções de suporte de dados vetoriais para formatos textuais padrão, como *Keyhole Markup Language* (KML), *Geography Markup Language* (GML), *GeoJSON*, *GeoHash* e *Well-known Text* (WKT) usando SQL', 'suporte para objetos 3D, índices espaciais e topologia de rede' [**21**].

2.2.2. *Firebase - Backend as a Service*

O *Firebase* é uma plataforma baseada em BaaS (*Backend as a Service*), ou seja, que gere e automatiza todo o *backend* e as funcionalidades básicas de uma aplicação tais como a autenticação de utilizadores, o armazenamento e a escalabilidade. Foi desenvolvido pela *Google* e facilita a criação de aplicações tanto móveis como *web*, através do recurso a API e SDK próprios [**22**].

Oferece duas soluções de base de dados baseadas em *cloud* com uma estrutura NoSQL e suporte de sincronização de dados em tempo real:

- O *Realtime Database* com uma estrutura em árvore de objetos JSON, onde os dados são armazenados em nós;
- O *Cloud Firestore* que guarda os dados em documentos, que por sua vez estão organizados em coleções. Os documentos podem conter objetos complexos (ex: dados do tipo *Map*, tipo de dados não relacionado a geografia, mas sim com uma estrutura de lista composta por elementos chave-valor) e subcoleções [**23**].

O *Realtime Database* é uma solução eficiente e de baixa latência. Em contrapartida, o *Cloud Firestore* apresenta um modelo de dados mais intuitivo, consultas mais avançadas e rápidas, e melhor escalabilidade que o *Realtime Database*.

O *Cloud Firestore* suporta dados do tipo *Array, Boolean, Bytes, Date* e *Time, Floating-point number, Geographical Point, Integer, Map, Null, Reference* e *Text string* [**24**].

Para a realização de consultas com base na localização o *Firebase* suporta a codificação de localizações geográficas.

2.2.3. Representação de Mapas

O bom funcionamento da aplicação exige a visualização de mapas. Segundo [**25**], as três soluções mais populares de código aberto para representação de mapas são: *Mapbox-GL*, *OpenLayers* e *Leaflet*.

Mapbox-GL

Mapbox-GL é uma biblioteca em *JavaScript* que renderiza mapas interativos através de *vector tiles*, ou blocos de vetor, um formato de dados leve para armazenar vetores de dados geoespaciais, como pontos, linhas e polígonos. Ao armazenar este tipo de dados, permite mapas de alta resolução, com rápido carregamento, armazenamento eficiente em *cache* e variados tipos de estilização como modo Diurno, Noturno, Estradas, Terreno e Satélite [**26**]. *Mapbox-GL* é uma ferramenta multiplataforma. A última versão permite o mapeamento 3D, céus personalizáveis, uma nova câmara e melhorias de desempenho [**27**].

Os dados geográficos têm origens diversas, sendo em parte propriedade do *Mapbox-GL*, pertencendo também a projetos de dados de acesso livre como o *OpenStreetMap* e determinadas áreas geográficas pertencem a vendedores de dados privados.

OpenLayers

OpenLayers é uma biblioteca de alto desempenho, código aberto e gratuita, que utiliza a linguagem *JavaScript*. Suporta serviços de mapeamento OGC e renderiza dados vetoriais de *GeoJSON*, *TopoJSON*, KML, GML, blocos de vetor *Mapbox* e outros formatos. É de fácil implementação, fácil extensão e multiplataforma [**28**].

Suporta dados geográficos de diferentes origens, entre os quais recorre ao *OpenStreetMap*, mapas da *Google*, *Mapbox*, *Stamen Design* e ESRI.

Leaflet

Leaflet é uma biblioteca em *JavaScript* e código aberto para mapas interativos compatíveis com dispositivos móveis. É uma ferramenta multiplataforma, de fácil implementação e extensão. Utiliza WMS, camadas de vetores: polilinha, polígono, círculos, retângulos, sobreposição de imagem e *GeoJSON*. Permite controlo sobre o mapa, ao nível de *zoom* ou escala, uma *performance* mais rápida, eliminação do atraso e navegação através do teclado [**29**].

OpenStreetMap

OpenStreetMap é um projeto global e gratuito de uma comunidade de mapeadores voluntários que contribuem e mantêm atualizados os dados sobre estradas, caminhos, cafés, estações ferroviárias e outro tipo de entradas por todo o mundo. O *OpenStreetMap* é constituído por dados abertos, desde que se utilizem corretamente os direitos de autor [**30**].

Mapas da *Google*

Com o *Maps Software Development Kit* (SDK) é possível adicionar mapas à aplicação pretendida com base nos dados presentes no *Google Maps*. A API gere automaticamente o acesso aos servidores do *Maps*, o descarregamento de dados, a exibição do mapa e a resposta a gestos no mapa. Permite também o manuseamento dos seguintes tipos de dados: ícones ancorados em posições específicas no mapa (marcadores), conjuntos de segmentos de linha (*polylines*), segmentos fechados (*polygons*), gráficos de *bitmap* ancorados em posições específicas no mapa (sobreposições de solo) e conjuntos de imagens que são exibidas sobre os blocos de mapas básicos (sobreposições de blocos) [**31**].

ArcGIS Server

ArcGIS Server é um servidor que fornece análises, recursos de mapeamento e recursos de gestão de dados ao sistema onde se encontra implementado. Suporta serviços da *Web* OGC e modelos de geoprocessamento personalizados. É oferecido em três edições: *Basic, Standard e Advanced*. Estas edições podem ser estendidas ou incorporam certas funcionalidades como: servir conjuntos de dados esquemáticos, partilhar serviços 3D e incorporar ferramentas de análise 3D, servir modelos geoestatísticos, partilhar ou incorporar ferramentas e serviços do *Spatial Analyst* (análises em tempo real) e suporte para mais de 385 formatos de dados, incluindo *Computer-Aided Design* (CAD), OGC, XML, JSON, *Really Simple Syndication* (RSS), *Database Management System* (DBMS) e *Comma-Separated Values* (CSV) [**32**].

Algumas fontes de dados são baseadas em arquivos, como arquivos CSV e XLS, ou baseadas em padrões abertos, como KML e OGC. Outras fontes de dados são nativas do *ArcGIS*.

2.3. Aplicações Relacionadas

Like a Local é uma plataforma que ajuda os utilizadores a encontrar lugares para visitar, baseando-se em opiniões de pessoas locais. Permite o acesso a listas de sugestões de percursos pré-criadas, a criação de listas baseadas nos comentários de outros utilizadores, a troca de mensagens com pessoas locais e a compra de experiências. Disponibiliza a visualização de mapas *offline* [**33**].

Esta aplicação utiliza como camada de mapas os dados do *OpenStreetMap*.

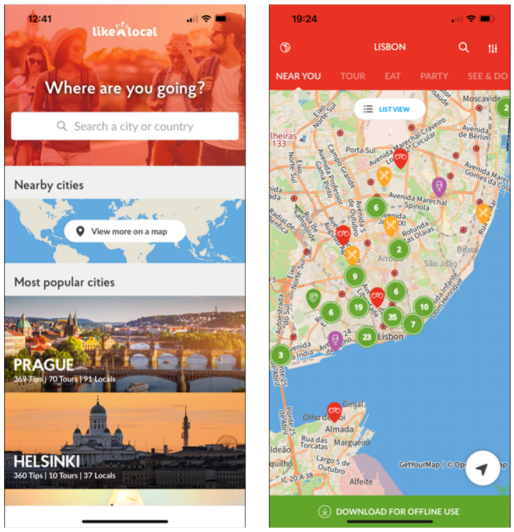


FIGURA 2. Aplicação *Like a Local*.

Visit a City é uma plataforma interativa com uma base de dados extensa, cobrindo múltiplas cidades à volta do mundo. Permite uma pesquisa de lugares com base em diferentes tipos de categorias que ajudam o utilizador a encontrar os melhores lugares a visitar e o acesso a um mercado de experiências extenso. Tem suporte de navegação *offline* [**34**]. Esta aplicação utiliza como base de visualização mapas da *Google*.

Roadtrippers é uma plataforma com uma base de dados extensa que cobre com mais detalhe o território da Austrália, do Canadá, da Nova Zelândia e dos EUA. Permite a criação de uma rota entre dois lugares, disponibilizando os pontos de interesse abrangidos pela mesma, a reserva de hotéis através da análise de fotografias e comentários, e uma pesquisa através de diferentes tipos de categorias com a possibilidade de guardar lugares como favoritos [**35**]. Esta aplicação utiliza como camada de mapas os dados do *OpenStreetMap*, do *Mapbox* e dados próprios.

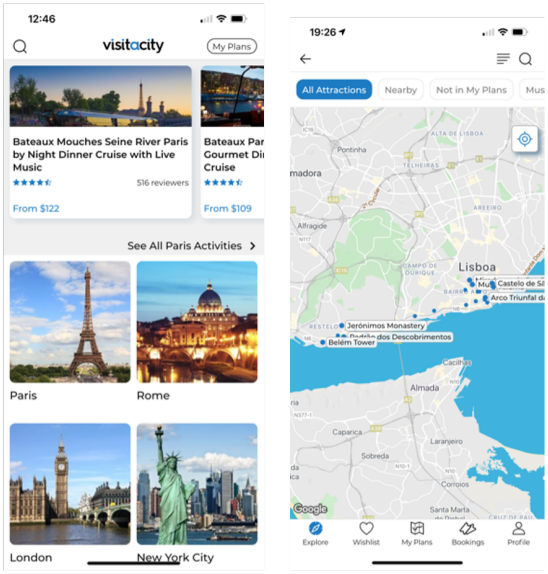


FIGURA 3. Aplicação *Visit a City*.

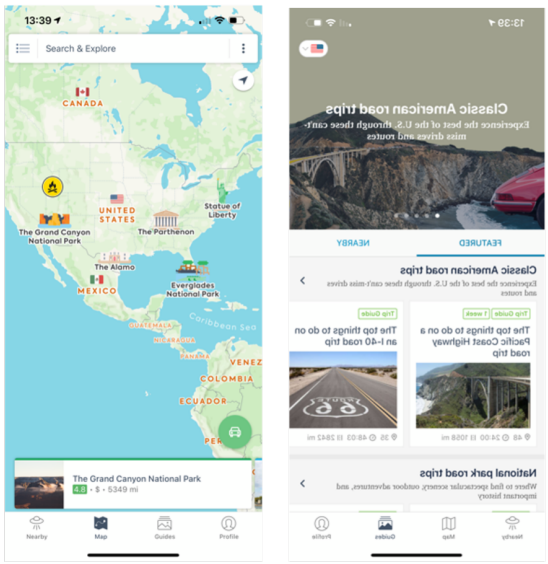


FIGURA 4. Aplicação *Roadtrippers*.

As aplicações apresentadas, apesar de terem informações partilhadas de fontes consideradas de confiança, principalmente a aplicação *Like a Local* por a partilha ser feita por indivíduos locais, carecem de algumas funcionalidades propostas na implementação da aplicação desenvolvida. Os marcadores no mapa limitam-se a pontos isolados ou pontos pertencentes a rotas, não existindo a possibilidade de seguir os seus criadores que muitas vezes, com interesses em comum, poderiam ser objeto de pesquisa para futuros lugares a visitar. Também não é possível a partilha dirigida de marcadores entre utilizadores, o que dificulta a difusão da informação.

Com a implementação desta aplicação procura-se aproximar o utilizador da informação e dos restantes utilizadores seus conhecidos, permitindo a criação de uma rede de amigos e de partilha de marcadores entre utilizadores.

CAPíTULO 3

Componentes e Ferramentas da Aplicação

Neste capítulo é apresentado o *Android Studio*, ambiente de desenvolvimento utilizado, e também os componentes da aplicação, com os respetivos detalhes.

3.1. Ambiente de Desenvolvimento

A implementação desta aplicação foi feita através do ambiente de desenvolvimento integrado da *Google* para *Android*, o *Android Studio*. Neste IDE é possível a implementação em linguagem *Java*, linguagem adotada neste projeto, ou *kotlin* [**36**].

A primeira versão do *Android* foi lançada em novembro de 2007 e desde então tem sido alvo de inúmeras atualizações com a natural adição de novas funcionalidades [**36**].

A aplicação do *Android Studio* disponibiliza informação relativa às várias versões disponíveis e à compatibilidade com os dispositivos móveis, como se observa na Figura 5. Neste projeto a versão mínima escolhida foi *Android 5.1 (Lollipop) API 22* e o nome do projeto foi *RoadBookProject*.

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99,8%
4.2 Jelly Bean	17	99,2%
4.3 Jelly Bean	18	98,4%
4.4 KitKat	19	98,1%
5.0 Lollipop	21	94,1%
5.1 Lollipop	22	92,3%
6.0 Marshmallow	23	84,9%
7.0 Nougat	24	73,7%
7.1 Nougat	25	66,2%
8.0 Oreo	26	60,8%
8.1 Oreo	27	53,5%
9.0 Pie	28	39,5%
10. Android 10	29	8,2%

FIGURA 5. Versões do *Android* e compatibilidade com dispositivos móveis.

3.2. Base de Dados

O sistema de informação utilizado nesta aplicação é o *Firebase*. Toda a informação, tanto a que é apresentada como a que é adicionada, encontra-se disponível remotamente e através da utilização da *Internet* pelos utilizadores que tenham efetuado o *login*.

De modo a atingir os objetivos propostos, foram integrados vários serviços desta ferramenta: *Firebase Authentication*, *Firebase Realtime Database*, *Cloud Firestore* e *Cloud Storage*.

O SDK do *Firebase Authentication* fornece métodos para criação e gestão de utilizadores. É necessário que a aplicação solicite as credenciais, de seguida são passadas ao SDK que efetua as devidas verificações enviando uma resposta ao utilizador. Após efetuado o *login* é possível ter acesso às informações do perfil e verificar a identidade do utilizador, através do *token* de autenticação [**37**].

O *Cloud Storage* é altamente escalável e os seus SDKs permitem guardar e exibir fotografias descarregadas diretamente do utilizador independentemente da qualidade da rede [**38**].

O *Cloud Firestore* permite armazenar dados no formato de *GeoPoint*, fator determinante neste projeto, pois um dos objetivos é guardar e trabalhar localizações num formato e com ferramentas adequados.

Informação adicional relativa ao *Firebase Realtime Database* e ao *Cloud Firestore* encontra-se no Capítulo 2.

3.2.1. Utilização *Offline*

Relativamente ao uso *offline*, tanto o *Cloud Firestore* como o *Firebase Realtime Database* são serviços compatíveis com a apresentação de dados *offline*.

Em relação ao *Cloud Firestore* é armazenada em *cache* uma cópia dos dados presentes no *Cloud Firestore*, sendo possível gravar, ler, procurar e consultar esses mesmos dados. Quando existe ligação à *Internet*, o *Cloud Firestore* sincroniza todas as alterações. A apresentação dos dados *offline* é gerida pela própria biblioteca do *Cloud Firestore* sem necessitar de implementação adicional [**39**].

Em relação ao *Firebase Realtime Database* é utilizado um recurso de persistência em disco. A aplicação grava os dados do *Firebase Realtime Database* localmente no dispositivo para poderem ser acedidos em caso de impossibilidade de comunicação com a rede [**40**].

Para que este recurso de persistência dos dados fosse utilizado em toda a aplicação foi necessária a criação de uma classe denominada *FirebaseHandler* com a linha de código correspondente, apresentada na Figura 6, e acrescentar ao ficheiro *Manifest* essa mesma classe, como apresentado na Figura 7.

```
public class FirebaseHandler extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        FirebaseDatabase.getInstance().setPersistenceEnabled(true);
    }
}
```

FIGURA 6. Classe *FirebaseHandler*.

```
<application
    android:name=".FirebaseHandler"
```

FIGURA 7. Ficheiro *Manifest*.

3.2.2. Integração com o *Android Studio*

Para integrar os SDKs do *Firebase* no *Android Studio* é necessário adicionar dependências e *plugins* no arquivo do *Gradle* no módulo.

A título de exemplo, a dependência relativa ao *Firebase Authentication* é exibida na Figura 8.

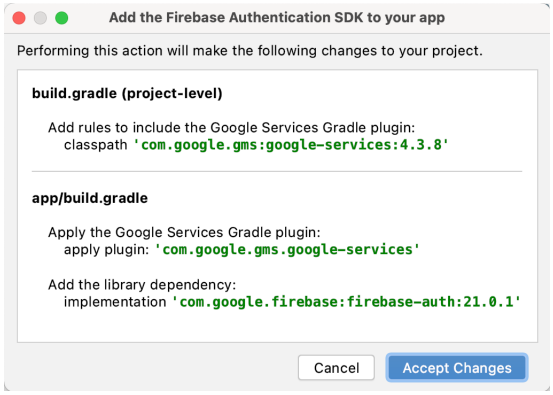


FIGURA 8. Dependência do SDK do *Firebase Authentication*.

As dependências relativas ao *Firebase Realtime Database*, ao *Cloud Firestore* e ao *Cloud Storage* são semelhantes.

3.3. Mapas e integração com o *Android Studio*

Nesta aplicação são utilizados os mapas da *Google* referidos na secção 2.2.3.

Para integrar o SDK dos mapas é necessário adicionar a respetiva dependência, exibida na Figura 9, e criar uma *Google Maps API key*, exibida na Figura 10.

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

FIGURA 9. Dependência do *Maps* SDK.

FIGURA 10. *Google Maps API key*.

Foram adotados os mapas da *Google* devido à compatibilidade e fluidez existente entre os dois serviços. Ao serem serviços pertencentes à *Google* a sua compatibilidade é significativa tanto ao nível de dependências, como de implementação dos diversos métodos e de grupos de esclarecimento de dúvidas. Por outro lado, existem também desvantagens: o facto de aparecerem marcadores irrelevantes e/ou de publicidade.

3.4. *Mockups*

Os *mockups* são entendidos como modelos de demonstração de um projeto à escala real ou não. Na Figura 11, é apresentado o modelo deste projeto realizado antes da implementação da aplicação como proposta da distribuição das várias funcionalidades e *designs*.

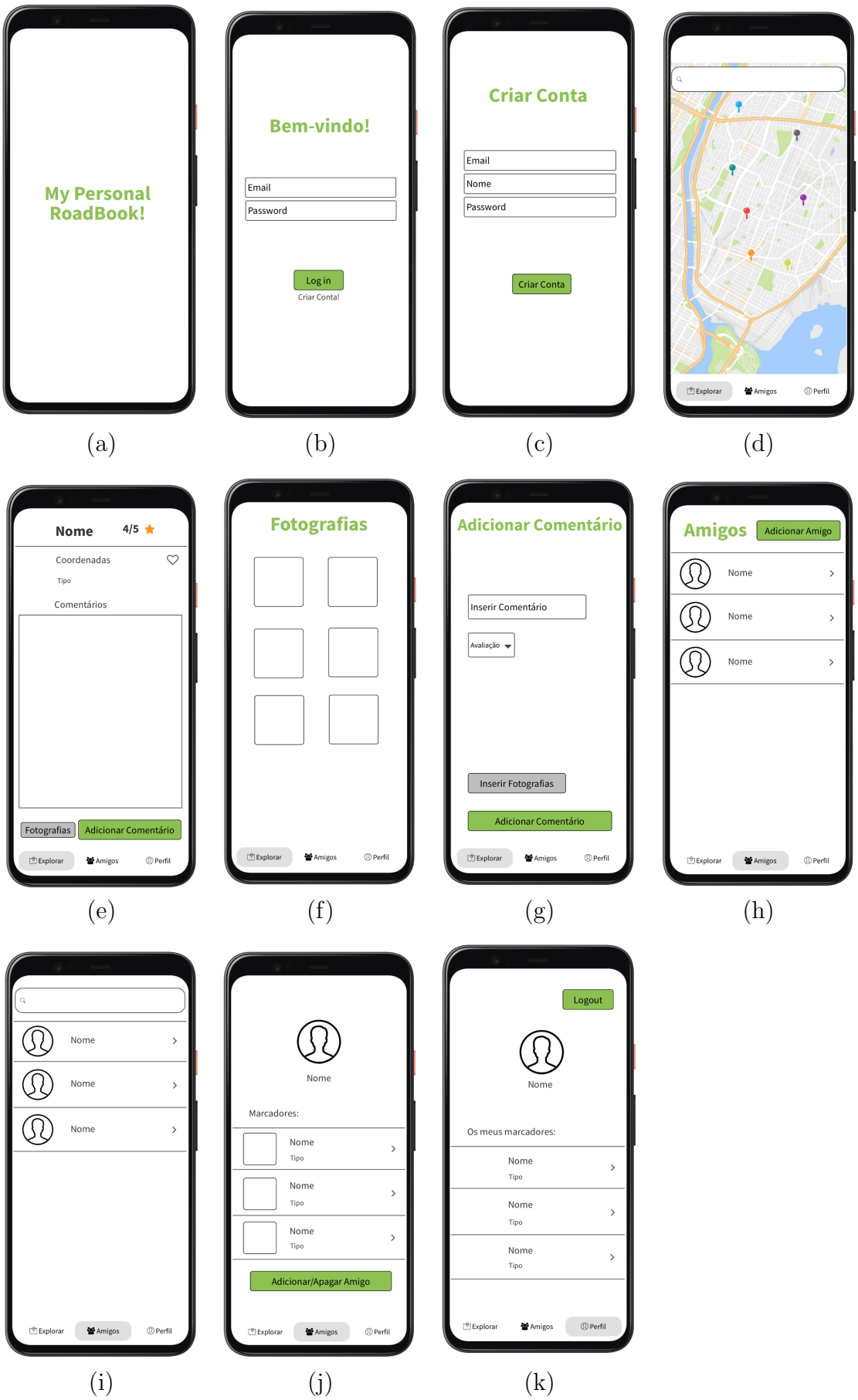


FIGURA 11. *Mockups da aplicação*. (a) *SplashScreen*; (b) *LogIn*; (c) Criar conta; (d) Separdor Explorar; (e) Informações do marcador; (f) Fotografias do marcador; (g) Adicionar coméntario ao marcador; (h) Separador *Amigos*; (i) Adicionar novo amigo; (j) Perfil do amigo; (k) Separador *Perfil*.

CAPíTULO 4

Estrutura da Aplicação

Para o desenvolvimento da aplicação foram utilizadas *atividades* (*activities*) e *fragmentos* (*fragments*). Uma *atividade* é entendida como um módulo único e independente onde são apresentadas as respetivas funcionalidades e corresponde, normalmente, a uma janela com UI. Cada *atividade* tem o seu ciclo de vida e os seus métodos [**41**]. Um *fragmento* representa o comportamento ou uma parte da UI, é entendido como uma secção modular de uma *atividade* com ciclo de vida próprio. Uma *atividade* suporta um conjunto de *fragmentos* que podem ser adicionados ou removidos durante a execução da mesma, os mesmos *fragmentos* podem ainda ser utilizados em atividades distintas [**42**].

4.1. Implementação

A criação da aplicação vai ser baseada na arquitetura apresentada nos diagramas da Figura 12 e 13, diagrama de componentes e diagrama de classes, respetivamente.

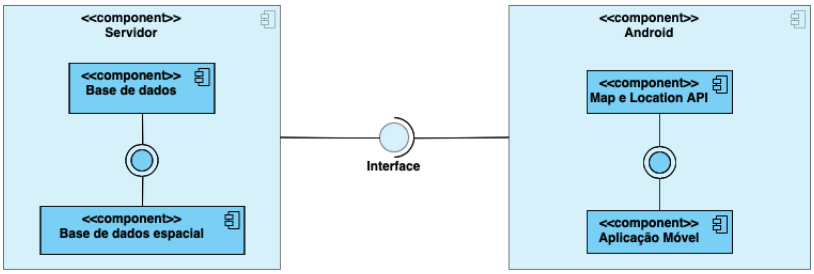


FIGURA 12. Diagrama de componentes sobre o funcionamento da aplicação.

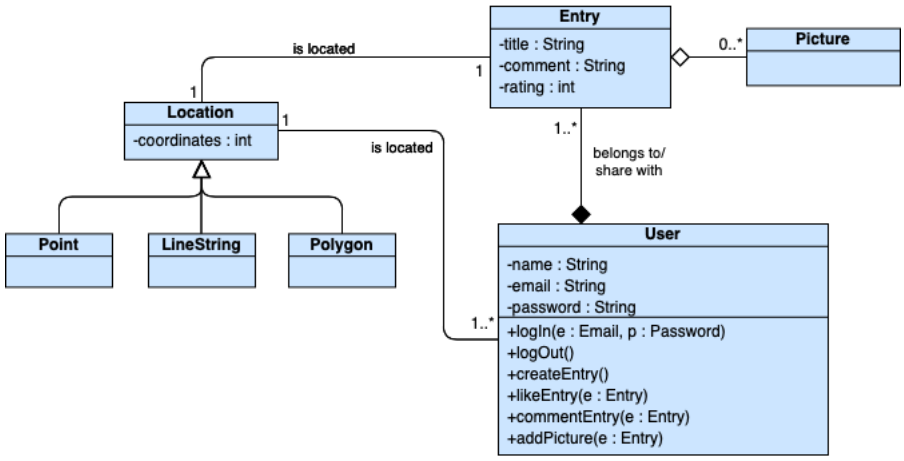


FIGURA 13. Diagrama de classes sobre a implementação da aplicação.

Os elementos que se verificam essenciais à criação da aplicação são:

- Base de dados espacial;
- Mapas para visualização de dados espaciais.

A integração destes elementos é possível através da implementação feita no ambiente de desenvolvimento.

De modo a facilitar a compreensão das tarefas permitidas ao utilizador é apresentado o seguinte diagrama de casos de uso:

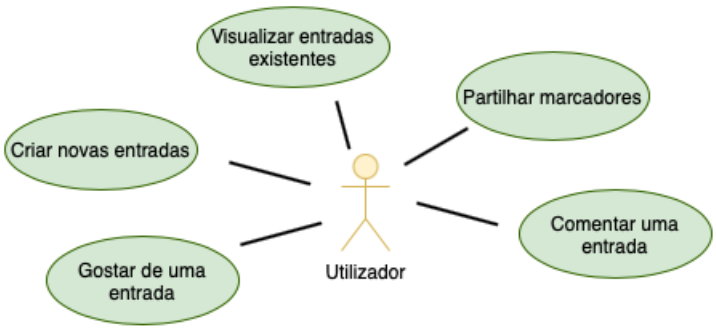


FIGURA 14. Casos de Uso.

Ao utilizador é permitido criar uma conta e fazer o respetivo *log in* e *log out*. É permitido visualizar entradas existentes, premindo os diferentes ícones presentes no mapa, e criar entradas acerca de um local preenchendo todos os campos do formulário respetivo, sendo necessário especificar a localização, adicionar fotografias ilustrativas, adicionar um comentário, adicionar uma avaliação e a privacidade do marcador. Outras tarefas disponibilizadas são: gostar e comentar a entrada de outro utilizador, clicando sobre o ícone respetivo presente no mapa, e a partilha de entradas com outros utilizadores.

A aplicação é constituída por 36 classes *Java* e respetivos *layouts*, das quais 13 são *fragmentos*, 5 são *atividades* e as restantes modelam *RecyclerViews*, como observado nas Figuras 15 e 16.

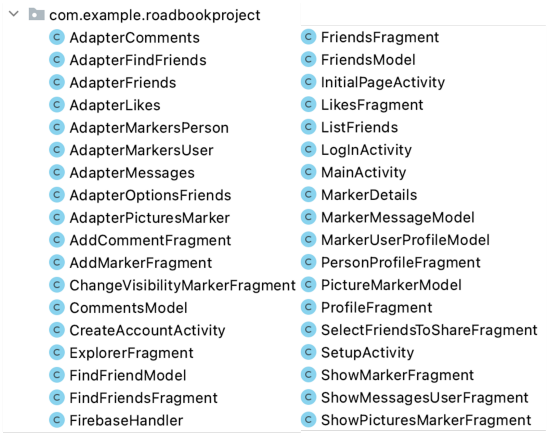


FIGURA 15. Classes *Java*.

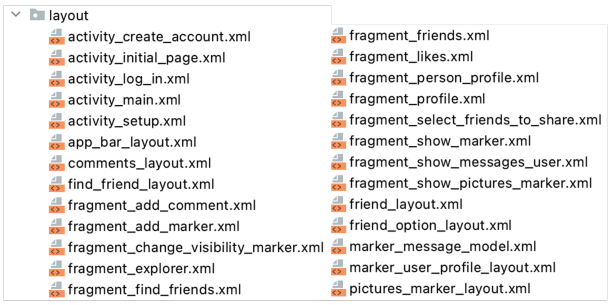
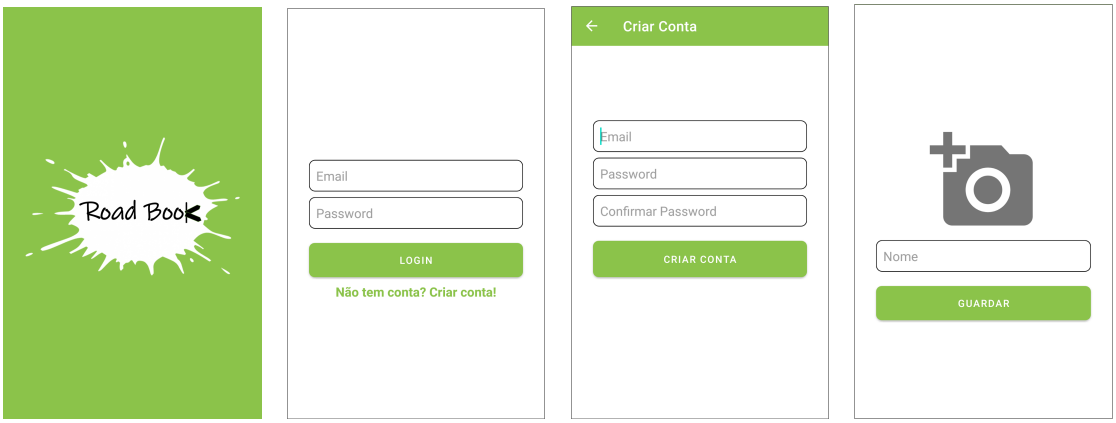


FIGURA 16. *Layouts*.

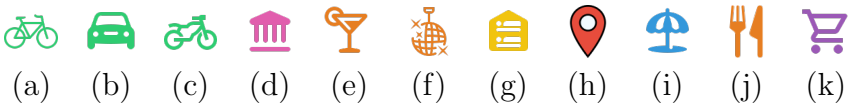
As Figuras que se seguem mostram a última versão da aplicação. O seu início é caracterizado pela apresentação de um *SplashScreen* (Figura 17 (a)), que é redirecionado para a *atividade* do *login* (Figura 17 (b)). Neste momento o utilizador pode iniciar sessão ou ser redirecionado para a *atividade* de criar conta (Figura 17 (c)), por premir o texto "Não tem conta? Criar conta!". Para a criação da nova conta é necessário preencher o *email*, a *password*, confirmar a *password* e completar a *atividade Setup* (Figura 17 (d)), onde são solicitados o nome do utilizador e uma fotografia de perfil, que surge após ser premido o botão *Criar Conta*.



(a) (b) (c) (d)

FIGURA 17. (a) SplashScreen, (b) LogIn, (c) Criar conta, (d) Conclusão da criação de conta.

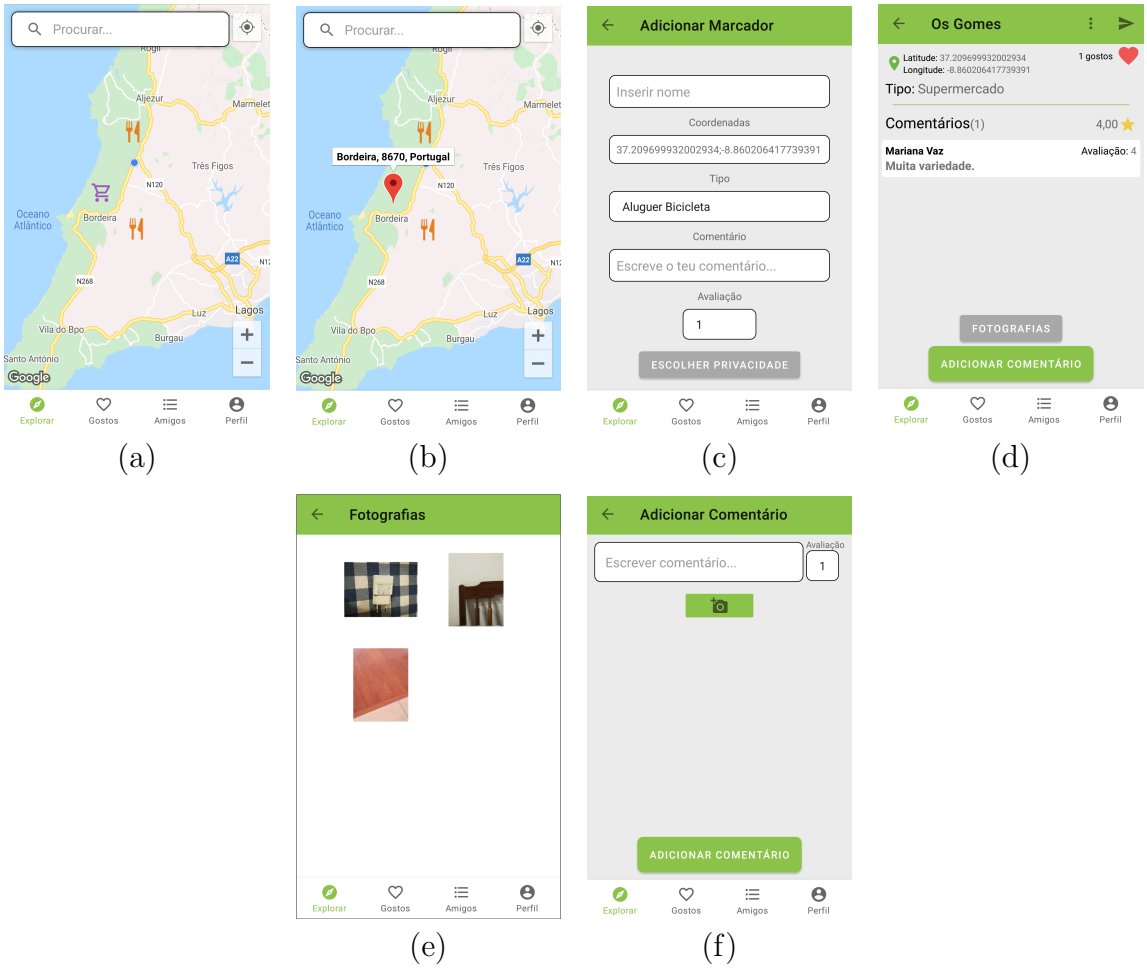
De seguida surge o *fragmento* Explorar (Figura 19 (a)), que corresponde à seleção do primeiro separador da barra de navegação, onde é apresentado um mapa com a localização do utilizador e todos os marcadores pertencentes ao ecrã visível presentes na base de dados aos quais o utilizador tem permissão para aceder. Cada marcador é apresentado com o correspondente ícone dependendo do seu tipo. A criação de uma categorização por tipo auxilia o utilizador a encontrar a entrada pretendida e torna a aplicação mais interativa. No topo do ecrã encontra-se uma barra de pesquisa, onde é permitido ao utilizador procurar por um lugar, facilitando assim a procura de comentários e fotografias nessa área. A categorização por tipo adotada inclui os seguintes tipos: *Aluguer Bicicleta, Aluguer Carro, Aluguer Mota, Atração/Monumento, Bar/Café, Discoteca, Hotel/Hostel, Outro, Praia, Restaurante* e *Supermercado*. A eventual adição de novos tipos à lista anterior seria um processo simples. Na Figura 18 observam-se os ícones associados a cada tipo de entrada.



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k)

FIGURA 18. Tipo de entradas: (a) Aluguer Bicicleta, (b) Aluguer Carro, (c) Aluguer Mota, (d) Atração/Monumento, (e) Bar/Café (f) Discoteca, (g) Hotel/Hostel, (h) Outro, (i) Praia, (j) Restaurante, e (k) Supermercado.

O utilizador consegue também adicionar um novo marcador (Figura 19 (c)), abrir um marcador para ver a sua informação (Figura 19 (d)) e fotografias respetivas (Figura 19 (e)), bem como adicionar um novo comentário (Figura 19 (f)) com fotografias descarregadas do seu dispositivo móvel.



(a) (b) (c) (d) (e) (f)

FIGURA 19. (a) Separador do mapa, (b) Início da criação de um marcador, (c) Adicionar marcador, (d) Informações do marcador, (e) Fotografias do marcador, (f) Adicionar comentário ao marcador.

Selecionando o segundo separador da barra de navegação, é apresentada a lista de marcadores *gostados* pelo utilizador (Figura 20), os quais podem ser pressionados para ser apresentada a respetiva informação.

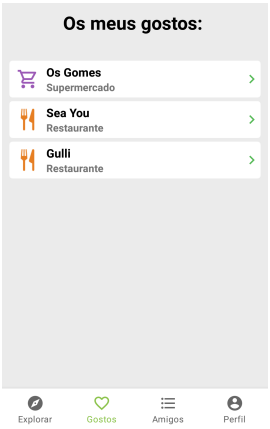
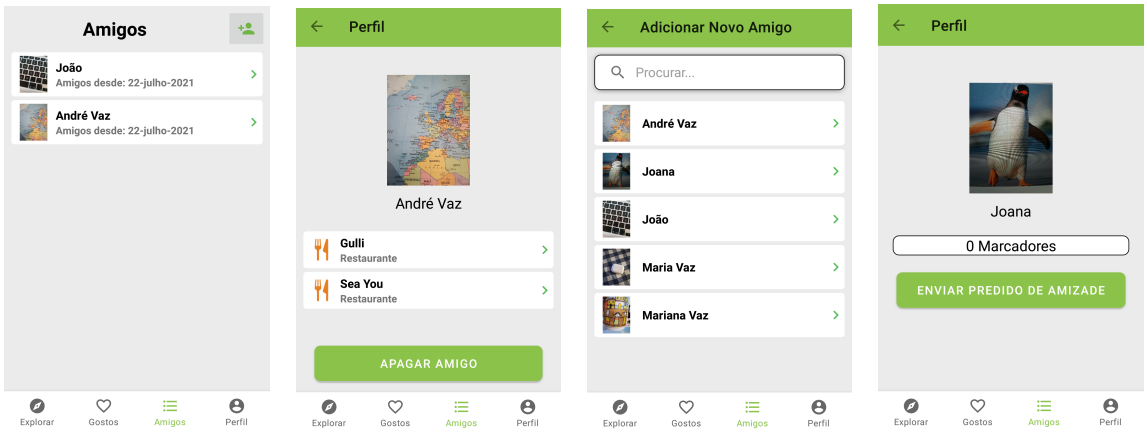


FIGURA 20. Separador dos gostos.

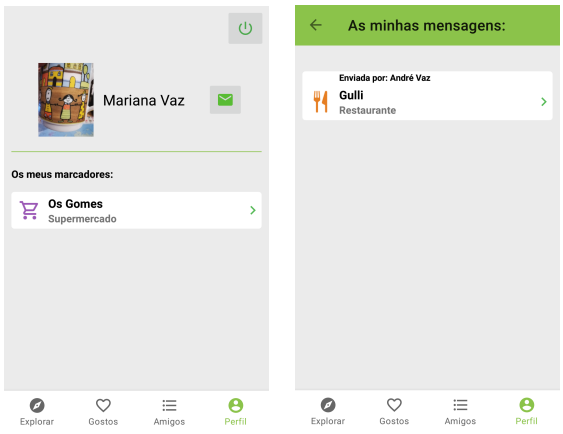
Selecionando o terceiro separador da barra de navegação, é apresentada a lista de amigos (Figura 21 (a)) do utilizador com um botão para adicionar novos amigos no topo. Quando selecionado um amigo da lista de amigos é apresentado o perfil do respetivo utilizador (Figura 21 (b)) com o nome, a fotografia e os marcadores criados pelo mesmo. Ao premir o botão para adicionar novos amigos, é apresentada uma lista com os utilizadores possíveis a adicionar como amigo (Figura 21 (c)). Quando aberto um elemento da lista, é apresentado o perfil do respetivo utilizador (Figura 21 (d)) com o botão de *Enviar Pedido de Amizade*.



(a) (b) (c) (d)

FIGURA 21. (a) Separador dos *Amigos*, (b) Perfil do amigo, (c) Adicionar novo amigo e (d) Perfil do utilizador a adicionar como amigo.

Selecionando o quarto separador da barra de navegação é exibido o perfil do utilizador (Figura 22 (a)), com o nome, a fotografia e marcadores respetivos. É ainda apresentado o botão de *logout* e o botão para aceder aos marcadores enviados por outros utilizadores que ao ser premido apresenta uma lista (Figura 22 (b)).



(a) (b)

FIGURA 22. (a) Separador do *Perfil* e (b) Marcadores recebidos.

4.2. *Firebase*

O *Firebase* tem uma estrutura dinâmica que facilita a implementação de leitura e escrita no sistema de informação.

O *Firebase Realtime Database* funciona com uma estrutura de nós, cada um com o seu próprio endereço, que permite a leitura e escrita de informação a partir da aplicação. Aos nós podem estar associados outros nós que contêm dados e estão organizados consoante uma estrutura hierárquica. Todas as ramificações possuem um endereço único que facilita a procura dos dados para leitura ou escrita [**22**]. Na Figura 23 (a) é apresentada a estrutura de nós do *Firebase Realtime Database* deste projeto.

O *Cloud Firestore* funciona com uma estrutura de coleções às quais estão associados documentos. Cada coleção tem um endereço único, bem como cada ficheiro pertencente às coleções. Na Figura 23 (b) é apresentada a estrutura de nós do *Cloud Firestore* deste projeto.

(a) (b) (c)

FIGURA 23. Estrutura das componentes do *Firebase*. (a) *Firebase Realtime Database*; (b) *Cloud Firestore*; (c) *Cloud Storage*.

Optou-se pela integração do *Firebase Realtime Database* e do *Cloud Firestore* em simultâneo para usufruir dos benefícios de ambos. Através do *Cloud Firestore* é possível

o armazenamento de dados em formato *GeoPoint* que não é possível no *Firebase Realtime Database*. Já o *Firebase Realtime Database* apresenta uma estrutura mais acessível em termos de programação e por existir há mais tempo apresenta um suporte para explicação de implementação e esclarecimento de dúvidas mais vasto, o que simplifica a sua implementação.

Sempre que existe a necessidade de aceder ao *Firebase Realtime Database* através do *Android Studio* é necessário recorrer ao método *FirebaseDatabase.getInstance.getReference()*, dando como argumento a referência ao nó. Como exemplo é apresentada a referência ao nó *users* através de *.child("users")*, ilustrado na Figura 24.

```
firebaseDatabase = FirebaseDatabase.getInstance("https://roadbookproject-7c206-default-rtdb.europe-west1.firebasedatabase.app/");
usersRef = firebaseDatabase.getReference().child("users");
```

FIGURA 24. Referência ao *Firebase Realtime Database*.

Para aceder ao *Firebase Authentication* é necessário recorrer ao método *FirebaseAuth.getInstance()*. No caso do *Cloud Storage* é necessário recorrer ao método *FirebaseStorage.getInstance().getReference().child()*, sendo necessário referenciar o nome de uma das coleções de fotografias. Por fim, no caso do *Cloud Firestore* é necessário recorrer ao método *FirebaseFirestore.getIntance().collection()*, sendo necessário referenciar o nome de uma das coleções presentes na base de dados. A Figura 25 ilustra um exemplo para cada uma das referências.

```
mAuth = FirebaseAuth.getInstance();
storageReference = FirebaseStorage.getInstance().getReference().child("profileImages");
firebaseFirestore = FirebaseFirestore.getInstance().collection( collectionPath: "markers");
```

FIGURA 25. Referência ao *Firebase Authentication*, ao *Cloud Storage* e ao *Cloud Firestore*, respetivamente.

4.2.1. Criar Conta e Autenticação

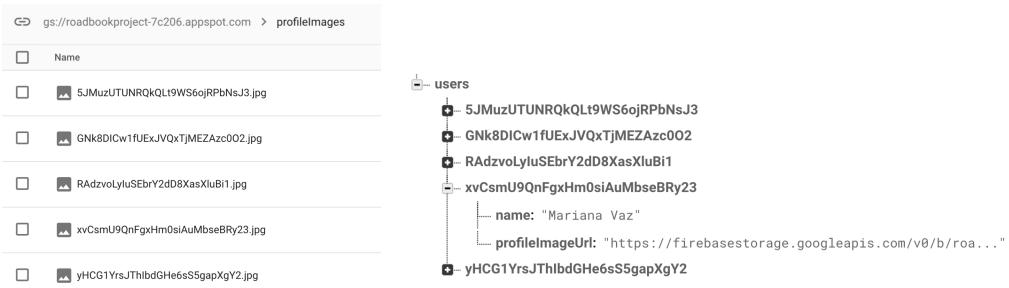
No *Firebase Authentication* está definido que a autenticação do utilizador é efetuada através do *email* e respetiva *password*, dados que ficam armazenados juntamente com o identificador de utilizador associado à conta, criados na Atividade *CreateAccount*, como ilustra a Figura 26, através do método *createUserWithEmailAndPassword(email, password)*. Não é verificada a existência do *email* do utilizador, mas é uma tarefa que se poderá realizar no futuro.

```
mAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull @NotNull Task<AuthResult> task) {
        if(task.isSuccessful()){
            Intent setupIntent = new Intent( packageContext: CreateAccountActivity.this, SetupActivity.class);
            setupIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
            startActivity(setupIntent);
            finish();
            progressDialog.dismiss();
        }else{
            Toast.makeText( context: CreateAccountActivity.this,  text: "Falha ao tentar criar conta.", Toast.LENGTH_SHORT).show();
            progressDialog.dismiss();
        }
    }
});
```

FIGURA 26. Método de criação de conta.

Para que a criação da conta prossiga ocorre uma verificação dos campos que lhe estão associados. É verificado se o *email* e a *password* estão preenchidos, se a *password* é superior a 6 dígitos, pois é um fator obrigatório deste método e se a *password* corresponde à confirmação da *password*.

Com a criação do primeiro utilizador surge o nó *users* e o respetivo nó com informação referente a esse mesmo utilizador, em que o endereço é o identificador devolvido aquando da criação da sua conta, garantindo-se assim a unicidade do mesmo. Sempre que surge um novo utilizador, o seu nó é associado ao nó *users*. Cada *user* contém o nome associado ao campo *name* e um *link* correspondente à fotografia de perfil guardada no diretório *profileImages* do *Cloud Storage* associado ao campo *profileImageUrl*, como observado na Figura 27 (a) e (b), respetivamente.



(a) (b)

FIGURA 27. (a) Fotografia de perfil no *Cloud Storage*; (b) *User* no *Firebase Realtime Database*.

A autenticação do utilizador é efetuada através da implementação do método *signInWithEmailAndPassword(email, password)* na Atividade *LogIn*. Para que a autenticação prossiga, na verificação dos campos que lhe estão associados é verificado se o *email* e a *password* estão preenchidos e se a *password* tem um comprimento superior a 6 dígitos.

4.2.2. Criar, Gostar e Comentar Marcador

Após a autenticação, o utilizador é redirecionado para o ecrã principal, onde é apresentado um mapa, a Atividade *InitialPage*, que assenta numa atividade composta por uma barra de navegação de rodapé e respetivos fragmentos.

No mapa e através de um clique longo sobre o ecrã é possível que o utilizador adicione um novo marcador, no Fragmento *AddMarker*. Com a criação do primeiro marcador surge a coleção *markers* com o respetivo documento com informação referente a esse mesmo marcador, sendo o endereço do documento gerado aleatoriamente. Sempre que surge um novo marcador, o seu documento é associado à coleção *markers*. Cada marcador contém uma *Geohash*, codificação de coordenadas geográficas, associada ao campo *geohash*, um *GeoPoint* associado ao campo *location*, um título associado ao campo *title*, um tipo associado ao campo *type*, uma *String* associada ao campo *visibility*, um *array* associado ao campo *friends* e o identificador do utilizador criador associado ao campo *userID*, como se observa na Figura 28.

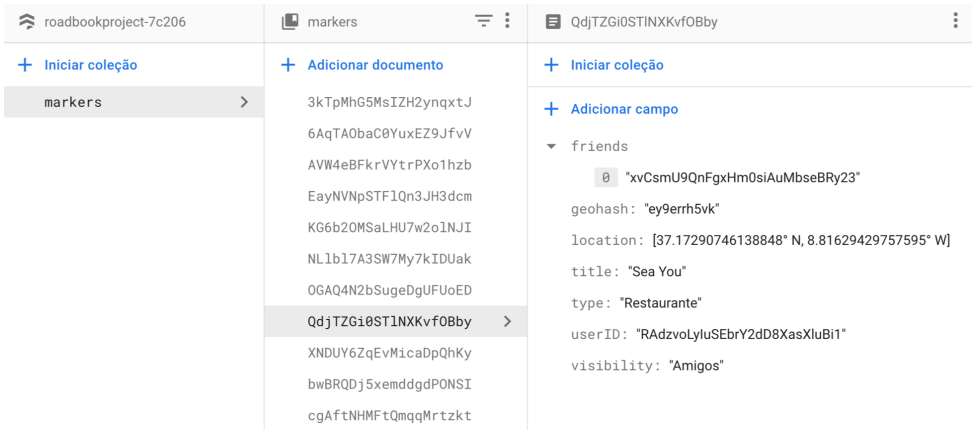


FIGURA 28. Marcador na coleção *markers*.

No mapa, através de um clique sobre um marcador é possível ver as informações relativas ao mesmo: o título, as coordenadas, o tipo, os comentários e os gostos, através do Fragmento *ShowMarker*. Com a adição do primeiro comentário surge o nó *comments* com um nó endereçado ao respetivo marcador, este por sua vez com um nó com informação referente a esse mesmo comentário. Sempre que surge um novo comentário, através do Fragmento *AddComment*, o seu nó é associado ao nó do marcador correspondente. Cada comentário contém um texto associado ao campo *comment*, uma avaliação quantitativa associada ao campo *rating* e o identificador do utilizador criador e respetivo nome associados aos campos *userID* e *user*, respetivamente como se observa na Figura 29.

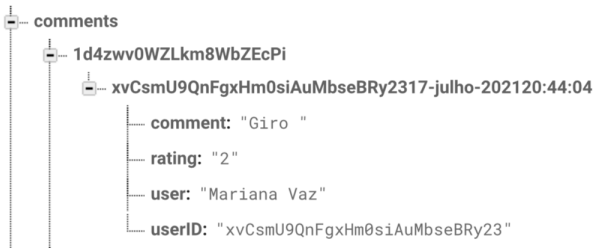


FIGURA 29. Comentário no nó *comments*.

Através do Fragmento *ShowMarker* é também possível ter acesso às fotografias do marcador clicando no botão Fotografias que redireciona o utilizador para o Fragmento *ShowPicturesMarker*. Com a adição da primeira fotografia surge o nó *imagesMarkers* com um nó endereçado ao respetivo marcador, este por sua vez com um nó com informação referente a essa mesma fotografia. Sempre que surge uma nova fotografia, o seu nó é associado ao nó do marcador correspondente. Cada fotografia contém um *link* correspondente à fotografia guardada no diretório *imagesCommentsMarkers* do *Cloud Storage* associado ao campo *markerImageUrl*, como observado na Figura 30 (a) e (b) respetivamente.

(a) (b)

FIGURA 30. (a) Fotografia no *Cloud Storage*; (b) Fotografia no nó *imagesMarkers*.

Através do Fragmento *ShowMarker* é ainda possível *gostar* do marcador através do ícone do coração ilustrado na Figura 31 (a). Com a adição do primeiro *gosto* surge o nó *likes* com um nó endereçado ao respetivo marcador, este por sua vez com um nó com informação referente a esse mesmo *gosto*. Sempre que surge um novo *gosto*, o seu nó é associado ao nó do marcador correspondente. Cada *gosto* tem como endereço o identificador do utilizador criador e contém apenas a palavra *true*, como observado na Figura 31 (b).

(a) (b)

FIGURA 31. (a) Ícone para *gostar* do marcador; (b) *Gosto* no nó *likes*.

A implementação do *gosto* é feita através de dois métodos:

- Um método em que é analisado na base de dados se o utilizador gosta do marcador em questão e caso se verifique é atribuído o número de *likes* e o coração do *gosto* fica preenchido a vermelho, senão é atribuído o número de *likes* e o coração do *gosto* fica sem preenchimento, como ilustrado na Figura 32.

```
databaseReference.child("likes").addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull @NotNull DataSnapshot snapshot) {
        if(snapshot.child(markerID).hasChild(currentUser)){
            countLikes = (int)snapshot.child(markerID).getChildrenCount();
            pressToLike.setImageResource(R.drawable.ic_like);
            showLikesNumber.setText(Integer.toString(countLikes) + " gostos");
        }else{
            countLikes = (int)snapshot.child(markerID).getChildrenCount();
            pressToLike.setImageResource(R.drawable.ic_dislike);
            showLikesNumber.setText(Integer.toString(countLikes) + " gostos");
        }
    }
    @Override
    public void onCancelled(@NonNull @NotNull DatabaseError error) {
    }
});
```

FIGURA 32. Método que atribui número de gostos e preenchimento do coração.

- Um método em que ao ser premido o ícone do coração é analisado na base de dados se o utilizador gosta do marcador em questão e caso se verifique esse *gosto* é removido da mesma, senão o *gosto* é adicionado, como ilustrado na Figura 33.

```
pressToLike.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        likeChecker = true;
        databaseReference.child("likes").addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull @NotNull DataSnapshot snapshot) {
                if(likeChecker.equals(true)){
                    if(snapshot.child(markerID).hasChild(currentUser)){
                        databaseReference.child("likes").child(markerID).child(currentUser).removeValue();
                        likeChecker = false;
                    }else{
                        databaseReference.child("likes").child(markerID).child(currentUser).setValue(true);
                        likeChecker = false;
                    }
                }
            }
            @Override
            public void onCancelled(@NonNull @NotNull DatabaseError error) {
            }
        });
    }
});
```

FIGURA 33. Método que adiciona/remove gosto à base de dados.

4.2.3. Privacidade dos Marcadores

Cada marcador criado e inserido na coleção *markers*, como referido na secção anterior, contém uma *String* associada ao campo *visibility*, não se optou por um enumerado pelo *Firebase Firestore* não suportar este tipo de dados, e um *array* associado ao campo *friends*. Estes dados dizem respeito à privacidade do marcador, restringindo a apresentação do marcador aos utilizadores desejados.

O campo *visibility* pode tomar um de quatro valores: *Público*, *Privado*, *Amigos* ou *Amigos Selecionados*, como apresentado na Figura 34. Quando *Público*, o marcador é apresentado a todos os utilizadores da aplicação. Quando *Privado*, o marcador é apresentado apenas ao utilizador que o criou. Quando é selecionado *Amigos*, o marcador é apresentado a todos os amigos do utilizador. Quando é selecionado *Amigos Selecionados*, surge a lista de amigos do utilizador, caso o utilizador tenha amigos, para selecionar os amigos aos quais permite que o marcador seja apresentado, como apresentado na Figura 35. Posteriormente e se desejado, o utilizador consegue acrescentar amigos aos amigos selecionados.

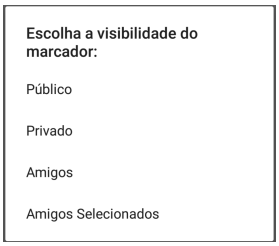


FIGURA 34. Opções de privacidade do marcador.

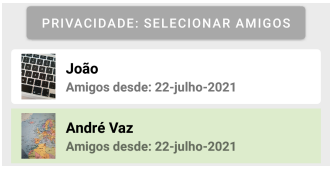


FIGURA 35. Lista de amigos a escolher.

4.2.4. Menu de opções na barra de topo

A apresentação do marcador, no Fragmento *ShowMarker*, é acompanhada por um menu de opções na barra de topo. O menu é composto por dois elementos, como ilustrado na Figura 36. O elemento da esquerda com o ícone de três pontos que desencadeia uma lista com duas opções: alterar a privacidade do marcador e apagar o marcador, como observado na Figura 37. O elemento da direita com o ícone da seta para a direita que permite partilhar o marcador com outro utilizador.

FIGURA 36. Menu de opções na barra de topo.

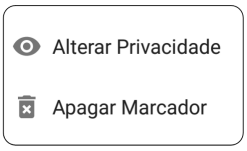


FIGURA 37. Opções do elemento da esquerda.

A implementação do menu de opções implica a criação de dois ficheiros, *menu_share_marker.xml* Figura 38 e *menu_item_option.xml* Figura 39, no diretório *menu* que são posteriormente invocados no Fragmento *ShowMarker*.

menu_share_marker.xml

```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/shareMarkerKey"
        app:showAsAction="always"
        android:icon="@drawable/ic_send"
        android:title="">
    </item>

</menu>
```

FIGURA 38. Ficheiro *menu_share_marker.xml*.

menu_item_options.xml

```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
        app:showAsAction="always"
        android:icon="@drawable/ic_options"
        android:title="">
        <menu>
            <item
                android:id="@+id/changeVisibilityOption"
                android:icon="@drawable/ic_visibility"
                android:title="Alterar Privacidade"
                app:showAsAction="never"/>
            <item
                android:id="@+id/deleteMarkerOption"
                android:icon="@drawable/ic_delete_forever"
                android:title="Apagar Marcador"
                app:showAsAction="never"/>
        </menu>

    </item>
</menu>
```

FIGURA 39. Ficheiro *menu_item_option.xml*.

Quando a visualização é feita pelo utilizador que criou o marcador são exibidos os dois elementos na barra de topo, caso o observador não seja o criador do marcador, não é exibido nenhum dos elementos. Esta gestão é feita através do recurso a uma *String* denominada *stateMenu* que pode tomar dois valores: *HIDE* e *SHOW*. É verificado o valor da *String stateMenu* e então apresentado o menu de opções, observar Figura 40 e 41 respetivamente.

```
if(documentSnapshot.getString( field: "userID").equals(currentUser)){
    stateMenu = "SHOW";
}else{
    stateMenu = "HIDE";
}
setHasOptionsMenu(true);
```

FIGURA 40. Verificação da *String stateMenu*.

```java
@Override
public void onCreateOptionsMenu(@NonNull @NotNull Menu menu, @NonNull @NotNull MenuInflater inflater) {
    if(stateMenu.equals("SHOW")) {
        inflater.inflate(R.menu.menu_item_options, menu);
        inflater.inflate(R.menu.menu_share_marker, menu);
    }
}
```

FIGURA 41. Apresentação do menu de opções.

Quando é selecionada a opção de alterar a privacidade do marcador, é apresentado o Fragmento *ChangeVisibilityMarker* constituído por dois botões: um relativo à seleção da nova privacidade e um para confirmar a seleção, como observado na Figura 42. Caso seja selecionada o tipo de privacidade *Amigos Selecionados*, surge a lista de amigos do utilizador para selecionar quem poderá ter acesso ao marcador. Para que as alterações fiquem guardadas, acede-se ao ficheiro do respetivo marcador e atualizam-se os campos *visibility* e *friends*.

FIGURA 42. Fragmento *ChangeVisibilityMarker*.

Quando é selecionada a opção de apagar o marcador, acede-se à coleção *markers*, aos nós *comments*, *likes* e *imagesMarkers* e à referência *imagesCommentsMarkers* da *Cloud Storage* para se remover os dados referentes ao marcador em questão. Esta implementação é observada na Figura 43.

```java
public void onClick(DialogInterface dialog, int which) {
    firebaseFirestore.collection( collectionPath: "markers").document(markerID).delete();
    databaseReference.child("comments").child(markerID).removeValue();
    databaseReference.child("likes").child(markerID).removeValue();
    databaseReference.child("imagesMarkers").child(markerID).get().addOnSuccessListener(new OnSuccessListener<DataSnapshot>() {
        @Override
        public void onSuccess(DataSnapshot dataSnapshot) {
            for(DataSnapshot ds: dataSnapshot.getChildren()){
                storageReference.child("imagesCommentsMarkers/" + ds.getKey() + ".jpg").delete();
            }
        }
    });
    databaseReference.child("imagesMarkers").child(markerID).removeValue();
    NavDirections action = ShowMarkerFragmentDirections.actionShowMarkerFragmentToExplorerFragment();
    Navigation.findNavController(view).navigate(action);
}
```

FIGURA 43. Implementação da remoção do marcador.

Quando é selecionada a opção de partilhar o marcador, surge o Fragmento *SelectFriendsToShare* com a lista de amigos do utilizador para serem selecionados os amigos a quem deseja enviar o marcador.

Associado a esta opção, existe no *Firebase Realtime Database* o nó *messages* com nós associados correspondentes aos destinatários de cada marcador enviado, em que os seus endereços são os identificadores desses mesmos utilizadores. Por cada marcador enviado, surge um nó com endereço criado aleatoriamente, é aleatório porque posteriormente não será necessário conhecer o endereço do nó para se lhe aceder, constituído por dois campos: o campo *fromUser* com o endereço do utilizador que enviou a mensagem e o campo *markerID* com o endereço do marcador enviado. Pode-se observar o nó *messages* e a correspondente implementação na Figura 45 e Figura 44, respetivamente.

```
for (String key: friendsSelected){
    HashMap<String,Object> message = new HashMap<~>();
    message.put("markerID", markerID);
    message.put("fromUser",currentUser);
    Calendar calForDate = Calendar.getInstance();
    SimpleDateFormat currentDate = new SimpleDateFormat( pattern: "dd-MMMM-yyyy");
    final String saveCurrentDate = currentDate.format(calForDate.getTime());
    SimpleDateFormat currentTime = new SimpleDateFormat( pattern: "HH:mm:ss");
    final String saveCurrentTime = currentTime.format(calForDate.getTime());
    final String randomId = currentUser + saveCurrentDate + saveCurrentTime;
    databaseReference.child(key).child(randomId).setValue(message);
}
```

FIGURA 44. Implementação do envio da mensagem.

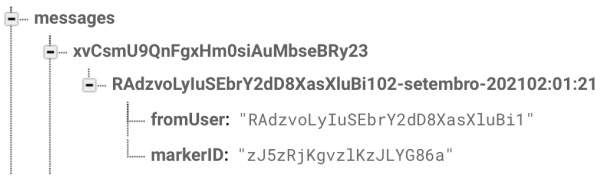


FIGURA 45. Nó *messages*.

A partilha do marcador permite ao utilizador recetor visualizar o marcador sem necessitar de o procurar no mapa. Por outro lado, quando é partilhado um marcador é verificada a sua privacidade. Caso a privacidade seja *Público* ou *Amigos*, não é realizada nenhuma alteração. Caso a privacidade seja *Privado*, a mesma é alterada para *Amigos Selecionados* sendo adicionado o utilizador recetor à lista de amigos autorizados a visualizar o marcador. Caso a privacidade seja *Amigos Selecionados*, é verificada a lista de amigos autorizados a visualizar o marcador e se o utilizador recetor não estiver na lista, é imediatamente adicionado.

4.2.5. Apresentar mensagens

Selecionando o quarto separador da barra de navegação de rodapé, o separador *Perfil*, é apresentado o Fragmento *Profile*. Se o utilizador premir o botão com o ícone de uma carta surge o Fragmento *ShowMessagesUser* que corresponde à exibição das mensagens

do utilizador. Neste fragmento é exibida a lista de marcadores enviados ao utilizador. Os marcadores correspondem ao campo *markerID* presente nos nós associados ao nó que tem como endereço o identificador do utilizador do perfil, este nó que por sua vez está associado ao nó *messages* presente no *Firebase Realtime Database*.

Cada mensagem apresenta o ícone, o nome e o tipo do marcador, bem como o nome do remetente da mensagem, como apresentado na Figura 46.

FIGURA 46. Estrutura da mensagem.

4.2.6. Adicionar amigo

Selecionando o terceiro separador da barra de navegação de rodapé, o separador *Amigos*, é apresentado o Fragmento *Friends*. Neste fragmento é exibida a lista de amigos do utilizador que corresponde aos nós associados ao nó *friends* presente no *Firebase Realtime Database*. Sempre que um pedido de amizade é aceite, são criados dois nós associados aos utilizadores envolvidos no pedido. Cada um deles tem como endereço o seu próprio identificador de utilizador e um nó associado referente ao utilizador com o qual a amizade foi estabelecida, com a data de início da amizade associada ao campo *date*, como observado na Figura 47.

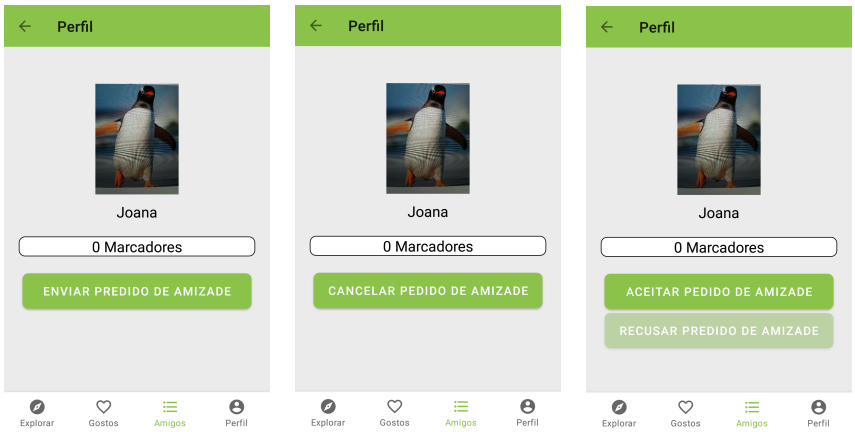
FIGURA 47. Amizade entre dois utilizadores no nó *friends*.

A partir deste fragmento e premindo o botão de adicionar amigo é apresentado o Fragmento *FindFriends*, onde é apresentada uma lista composta por todos os utilizadores da aplicação. Ao selecionar um utilizador é apresentado o seu perfil, Fragmento *PersonProfile*, e um botão de envio do pedido de amizade. Por outro lado, no recetor do pedido de amizade, é necessário que o utilizador percorra o mesmo caminho até ao perfil do utilizador que enviou o pedido e selecione *Aceitar*, para serem então adicionados os nós necessários ao nó *friends* e a amizade estabelecida.

Para que esta implementação seja possível é necessário recorrer ao nó *friendsRequests*, observado na Figura 48, onde os nós adicionados dizem respeito ao estado do pedido de amizade e aos respetivos emissor e destinatário. São utilizados como endereço os identificadores desses utilizadores, sendo atribuído um valor ao campo *requestType*. Estes estados são avaliados e deles depende a apresentação dos botões de enviar, cancelar, aceitar e recusar pedido de amizade quando o perfil de um utilizador é selecionado, como ilustra a Figura 49.

Num cenário ideal, o utilizador poderia escolher não figurar no diretório público de modo a evitar *spam*.

FIGURA 48. Pedido de amizade entre dois utilizadores no nó *friendsRequests*.



(a) (b) (c)

FIGURA 49. Aparência do perfil do utilizador. (a) Botão enviar pedido de amizade; (b) Botão cancelar pedido de amizade; (c) Botões aceitar e recusar pedido de amizade.

A implementação do pedido de amizade é feita através de três métodos:

- *sendFriendRequest()*, onde são adicionados no nó *friendsRequests* o respetivo emissor e destinatário do pedido de amizade e é também alterada a variável *CURRENT_STATE* para *request.sent*. Pode observar-se o código da implementação na Figura 50. O estado da variável *CURRENT_STATE* é encarregue pelo texto apresentado nos botões presentes no perfil dos utilizadores envolvidos no pedido de amizade, sendo através do método *MaintenanceOfButtons()* que essa gestão é feita.

```
private void sendFriendRequest() {
    friendRequestRef.child(senderUser).child(receiverUser).child("requestType").setValue("sent")
            .addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull @NotNull Task<Void> task) {
            if(task.isSuccessful()){
                friendRequestRef.child(receiverUser).child(senderUser).child("requestType").setValue("received")
                        .addOnCompleteListener(new OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull @NotNull Task<Void> task) {
                        if(task.isSuccessful()){
                            personProfileSendRequest.setEnabled(true);
                            CURRENT_STATE = "request_sent";
                            personProfileSendRequest.setText("Cancelar Pedido de Amizade");
                            personProfileDeclineRequest.setVisibility(View.INVISIBLE);
                            personProfileDeclineRequest.setEnabled(false);
                        }
                    }
                });
            }
        }
    });
}
```

FIGURA 50. Implementação do método *sendFriendRequest()*.

- *cancelFriendRequest()*, onde são removidos no nó *friendsRequests* o respetivo emissor e destinatário do pedido de amizade e é também alterada a variável *CURRENT_STATE* para *not_friends*.
- *acceptFriendRequest()*, onde são adicionados ao nó *friends* os nós referidos no início desta secção, bem como removidos no nó *friendsRequests* o respetivo emissor e destinatário do pedido de amizade, alterando-se a variável *CURRENT_STATE* para *friends*.

Após ser estabelecida uma amizade, é sempre possível eliminar um amigo através do método *unfriendExistingFriend()* onde são removidos do nó *friends*, os nós respetivos aos utilizadores envolvidos na amizade e a variável *CURRENT_STATE* é alterada para *not.friends*.

No Fragmento *PersonProfile* existem apenas dois botões com as funções de enviar, cancelar, aceitar e recusar o pedido de amizade e apagar o amigo. Para que não seja necessário a existência de um botão para cada função foi implementado o método *maintenanceOfButtons()* encarregue de gerir o texto apresentado nos botões bem como tornar ou não visível a apresentação dos botões, através da análise entre o perfil do utilizador e o perfil que o utilizador abre. Através deste método é analisado o nó *friendsRequests* e, caso existam nós associados aos utilizadores envolvidos, é analisado o estado do campo *requestType*. Caso o resultado seja *sent* a variável *CURRENT_STATE* é alterada para *request_sent*, o primeiro botão apresenta o texto "Cancelar Pedido de Amizade"e a visibilidade do segundo botão é negada. Caso o resultado do campo *requestType* se verifique *received* a variável *CURRENT_STATE* é alterada para *request_received*, o primeiro botão apresenta o texto "Aceitar Pedido de Amizade"e a visibilidade do segundo botão é permitida. Se não existirem nós associados aos utilizadores envolvidos é analisado o nó *friends*

e caso verifique a amizade entre os mesmos utilizadores a variável *CURRENT_STATE* é alterada para *friends*, o primeiro botão apresenta o texto "Apagar Amigo"e a visibilidade do segundo botão é negada.

4.3. Ficheiro *AndroidManifest.xml*

Este ficheiro é obrigatório e gerado automaticamente em qualquer projeto, pois descreve informações essenciais às ferramentas de compilação do *Android*: o nome do pacote da aplicação que determina o local das entidades do código e será o identificador exclusivo da aplicação no sistema e no *Google Play*; os componentes da aplicação, incluindo serviços e atividades; as permissões necessárias para aceder a zonas protegidas do sistema ou de outras aplicações; e os recursos de *hardware* e *software* exigidos pela aplicação que limitam os dispositivos compatíveis [**43**].

Para o bom funcionamento da aplicação foram adicionadas as permissões apresentadas na Figura 51, que permitem o acesso à *Internet*, acesso à localização precisa e o acesso à câmara.

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

<uses-feature android:name="android.hardware.camera"
    android:required="true" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="18" />
```

FIGURA 51. Permissões adicionadas.

4.4. Ecrã de abertura da aplicação

O *SplashScreen* ou ecrã de abertura é o ecrã com que o utilizador se depara ao abrir a aplicação, na Atividade *Main*. É constituído por uma imagem, um ficheiro de animação de entrada e a respetiva implementação através de um *Handler()* com duração de 3 segundos. Após este ecrã, é verificado se o utilizador está autenticado para que na eventualidade da aplicação ser fechada e novamente aberta não seja necessária uma nova autenticação, mas sim um redirecionamento do utilizador para o mapa.

4.5. Codificação *geohash*

Sendo esta uma aplicação com a apresentação de mapa, temos uma coleção de marcadores para apresentar. De modo a tornar esta apresentação mais leve foi implementado um sistema baseado em codificação de localizações geográficas, também conhecido como codificação *geohash*, oferecido pelo *Firebase*.

Uma *geohash* é a codificação de coordenadas geográficas (latitude e longitude) em *strings* de números e letras que correspondem a uma área no mapa, que se denomina célula. Quanto mais caracteres tiver a *string*, mais precisa será a localização [**44**].

As *geohashes* utilizam um sistema de codificação alfabético de Base32. Imaginemos o mundo dividido numa grelha com 32 células. O primeiro caractere de cada *geohash* corresponde à localização de uma das 32 células iniciais. Esta célula, por sua vez será

composta também por 32 células e por aí adiante. Ao acrescentar um caractere à *geohash* ocorre uma subdivisão do mapa, ou seja, ocorre um *zoom* na área sendo apresentada com maior detalhe [**44**]. A Figura 52, que usa uma Base4, ajuda a compreensão da explicação anterior.

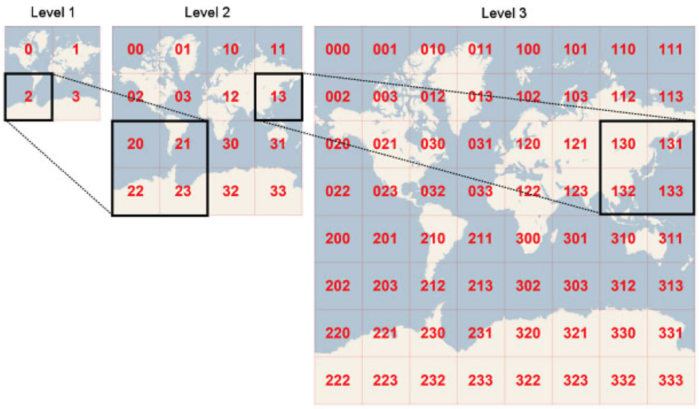


FIGURA 52. Divisão do mundo em células.

'O fator de precisão determina o tamanho da célula. Por exemplo, um fator de precisão de um cria uma célula de 5.000 km de altura e 5.000 km de largura, um fator de precisão de seis cria uma célula de 0,61 km de altura e 1,22 km de largura, e um fator de precisão de nove cria uma célula de 4,77 m de altura e 4,77 m de largura (as células nem sempre são quadradas).' Referido em [**44**].

As *geohashes* que integram os marcadores da aplicação desenvolvida são compostas por dez caracteres, o que significa que têm um fator de precisão de dez.

4.5.1. Biblioteca Geográfica

O *GeoFire* é uma biblioteca para Android que permite guardar e consultar um conjunto de chaves com base na sua localização. Uma das implementações é o *GeoFireUtils* que disponibiliza um conjunto de funcionalidades que facilitam a criação de *geoqueries* na aplicação, bem como a sua integração com o *Cloud Firestore* [**45**].

Para integrar o *GeoFire* no *Android Studio* é necessário recorrer à dependência presente na Figura 53.

```
implementation 'com.firebase:geofire-android-common:3.1.0'
```

FIGURA 53. Dependência do *GeoFire*.

No Fragmento *AddMarker* existe a criação de um marcador por parte do utilizador. Este marcador baseia-se numa estrutura, a classe *Java MarkerDetails*, composta pelos seguintes atributos: um *GeoPoint* denominado *location*, uma *String* denominada *title*, uma *String* denominada *userID*, uma *String* denominada *type*, uma *String* denominada *geohash*, uma *String* denominada *visibility* e um *ArrayList* denominado *friends*. O atributo

geoHash é criado com base na localização atribuída ao marcador, através do método *GeoFireUtils.getGeoHashForLocation()*. Este *MarkerDetails* é adicionado à coleção *markers*, como observado na Figura 54.

```
GeoPoint geoPoint = new GeoPoint(latitude,longitude);
String hash = GeoFireUtils.getGeoHashForLocation(new GeoLocation(geoPoint.getLatitude(), geoPoint.getLongitude()));
MarkerDetails markerDetails = new MarkerDetails(geoPoint,nameMarker,currentUser,typeSelected,
        hash, visibility, friendsSelected);
firebaseFirestore.collection( collectionPath: "markers").add(markerDetails)
```

FIGURA 54. Criação da GeoHash e adição do *MarkerDetails* à coleção *markers*.

Posteriormente, no Fragmento *Explorer* é apresentado o mapa e exibidos os marcadores armazenados na coleção *markers* através do método *showMarkersOnMapOnMoveCamera()*. De modo a não sobrecarregar a aplicação e os serviços, são apenas descarregados do *Firebase* os marcadores pertencentes à região visível no ecrã, considerando a privacidade associada a cada um deles.

As *geohashes* de todos os marcadores são analisadas através do método *GeoFireUtils.getGeoHashQueryBounds()*, que tem como argumentos um ponto central e uma distância de raio. O cálculo da região visível tem em conta estes dois argumentos. Primeiramente obtém-se a região visível através do método *getVisibleRegion()* disponibilizado pela biblioteca *GeoFire*. Através do método *getCenter()* encontra-se o ponto central da região visível. A região visível providencia quatro variáveis *farRight*, *farLeft*, *nearRight*, *nearLeft*, que correspondem ao canto superior direito, ao canto superior esquerdo, ao canto inferior direito e ao canto inferior esquerdo do ecrã, respetivamente. Fez-se uso das variáveis *farLeft* e *nearRight* para que através do uso do método *Location.distanceBetween()* se calcule a diagonal do ecrã. Ao dividir essa diagonal por dois descobre-se a distância de raio a utilizar, como se observa na Figura 55.

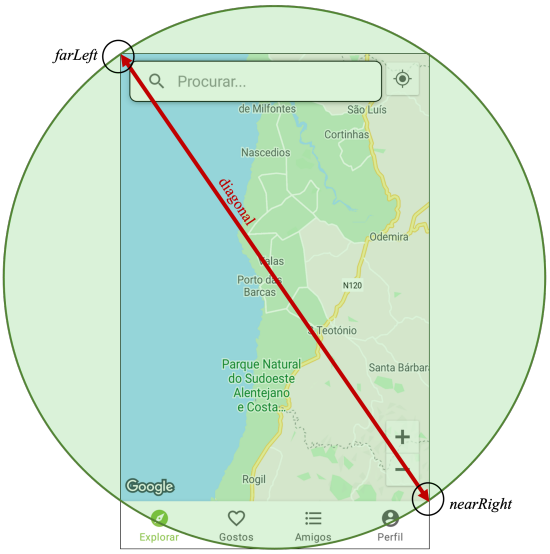


FIGURA 55. Ilustração da distância de raio do ecrã.

Após a análise de todas as *geohashes* e devido à precisão da *GeoHash* é necessário filtrar falsos positivos, conforme sugerido em "Consultas com base na localização"[**46**]. É

atribuído a cada marcador o ícone correspondente ao seu tipo, através do método *chooseIcon()*, e analisada a sua privacidade, através do método *setPrivacyMarker()*. A implementação do método *showMarkersOnMapOnMoveCamera()* é apresentada na Figura 56, 57 e 58.

```
private void showMarkersOnMapOnMoveCamera() {
    VisibleRegion visibleRegion = map.getProjection().getVisibleRegion();
    LatLng centerVisibleScreen = visibleRegion.latLngBounds.getCenter();
    final GeoLocation center = new GeoLocation(centerVisibleScreen.latitude, centerVisibleScreen.longitude);
    LatLng farLeft = visibleRegion.farLeft;
    LatLng nearRight = visibleRegion.nearRight;
    float[] diagonalDistance= new float[1];
    Location.distanceBetween(
            farLeft.latitude,
            farLeft.longitude,
            nearRight.latitude,
            nearRight.longitude,
            diagonalDistance);
    final double radiusInM = diagonalDistance[0] / 2;
    List<GeoQueryBounds> bounds = GeoFireUtils.getGeoHashQueryBounds(center, radiusInM);
    final List<Task<QuerySnapshot>> tasks = new ArrayList<>();
    for (GeoQueryBounds b : bounds) {
        Query q = referenceMarkers
                .orderBy("geohash")
                .startAt(b.startHash)
                .endAt(b.endHash);
        tasks.add(q.get());
    }
```

FIGURA 56. Método que adiciona os marcadores presentes na coleção *markers* ao mapa. Verifica-se a região visível e são ordenadas as *geohashes* consoante a distância à região visível.

```
Tasks.whenAllComplete(tasks)
        .addOnCompleteListener(new OnCompleteListener<List<Task<?>>>() {
            @Override
            public void onComplete(@NonNull Task<List<Task<?>>> t) {
                List<DocumentSnapshot> matchingDocs = new ArrayList<>();
                for (Task<QuerySnapshot> task : tasks) {
                    QuerySnapshot snap = task.getResult();
                    for (DocumentSnapshot doc : snap.getDocuments()) {
                        GeoPoint geoPoint = doc.getGeoPoint( field: "location");
                        double lat = geoPoint.getLatitude();
                        double lng = geoPoint.getLongitude();
                        GeoLocation docLocation = new GeoLocation(lat, lng);
                        double distanceInM = GeoFireUtils.getDistanceBetween(docLocation, center);
                        if (distanceInM <= radiusInM) {
                            matchingDocs.add(doc);
                        } } }
```

FIGURA 57. Continuação do método que adiciona os marcadores presentes na coleção *markers* ao mapa. Verificam-se as *geohashes* que se encontram presentes na região visível para serem apresentados no mapa.

```
List<Marker> markersOnMapVisible = new ArrayList<~>();
for(DocumentSnapshot resultSnapshot: matchingDocs){
    MarkerOptions markerOptions = new MarkerOptions().position(new LatLng(resultSnapshot
            .getGeoPoint( field: "location").getLatitude(), resultSnapshot
            .getGeoPoint( field: "location").getLongitude())).title(resultSnapshot.getString( field: "title"));
    chooseIcon(resultSnapshot, markerOptions);
    setPrivacyMarker(resultSnapshot, markerOptions, markersOnMapVisible);
}
for(Marker mkr: markersOnMap){
    if(!markersOnMapVisible.contains(mkr)){
        mkr.remove();
    } }
```

FIGURA 58. Continuação do método que adiciona os marcadores presentes na coleção *markers* ao mapa. Às *geohashes* presentes na região visível faz-se corresponder um marcador com o devido ícone e analizada a sua privacidade.

4.6. Mapas da *Google*

A utilização dos mapas da *Google* é feita através da interação entre a classe *SupportMapFragment* e a *interface OnMapReadyCallback*, oferecidas pelo *Android*.

Através da classe *SupportMapFragment*, é criado um fragmento mapa que quando sincronizado pelo método *getMapAsync()* define um objeto *callback* que será acionado quando a instância *GoogleMap* estiver pronta para ser usada. Este objeto *callback* instanciado pela *interface OnMapReadyCallback* implica a implementação do método *onMapReady()* que será invocado quando a instância *GoogleMap* referida estiver pronta para ser usada. Todas as implementações possíveis ao mapa são feitas no método *onMapReady()*. A implementação é ilustrada na Figura 59.

```
if (map == null) {
    SupportMapFragment mapFragment = (SupportMapFragment) getChildFragmentManager()
            .findFragmentById(R.id.mapView);
    mapFragment.getMapAsync( callback: this);
}
```

FIGURA 59. Implementção da classe *SupportMapFragment* no Fragmento *Explorer*

4.7. *Interface SharedPreferences*

Quando o Fragmento *Explorer* onde se encontra o mapa é interrompido, a localização apresentada é guardada através da *interface SharedPreferences*. Para que, quando selecionado novamente esse separador na barra de navegação de rodapé, o mapa seja exibido na mesma posição e o utilizador não perca a sua navegação. Quando o fragmento é interrompido, a localização apresentada é guardada no método *onStop()*, através da criação de um elemento *CameraPosition* relativo à posição da câmara na altura da interrupção e à criação de um elemento *SharedPreferences* onde se referencia a latitude e longitude dessa mesma posição. Para a localização ser exibida, existe uma implementação da *interface* no método *onMapReady()*, verificam-se a latitude e a longitude guardadas no elemento *SharedPreferences* e move-se a câmara do mapa para a posição criada através dos elementos guardados.

4.8. Geocodificação e Geocodificação Inversa

O *Android* disponibiliza a classe *Geocoder* para lidar com Geocodificação (*Geocoding*) e Geocodificação Inversa (*Reverse Geocoding*). Entende-se por Geocodificação o processo de transformar um endereço no mapa em coordenadas geodésicas. E por Geocodificação Inversa o processo inverso, transformar coordenadas geodésicas num endereço no mapa [**47**].

Neste projeto fez-se uso desta funcionalidade em algumas circunstâncias.

O mapa apresentado no fragmento *Explorer* contém uma barra de pesquisa, em que o utilizador pode pesquisar pelo nome de uma localização com a câmara a alterar a sua posição para essa mesma localização. Esta implementação é possível através do método *getFromLocationName()* da classe *Geocoder*. O nome pesquisado pelo utilizador é extraído e está associado a um par de coordenadas para o qual a câmara do mapa é redirecionada. Pode-se observar a implementação desta funcionalidade na Figura 60.

```
search_bar_onMap.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
    @Override
    public boolean onQueryTextSubmit(String query) {
        String location = search_bar_onMap.getQuery().toString();
        List<Address> addressList = null;
        try {
            addressList = geocoder.getFromLocationName(location, maxResults: 1);
            if(addressList.isEmpty()){
                Toast.makeText(getContext(), text: "A localidade não é válida, tente outra vez.", Toast.LENGTH_SHORT).show();
            }else {
                Address address = addressList.get(0);
                LatLng latLng = new LatLng(address.getLatitude(), address.getLongitude());
                map.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, zoom: 15));
            }
        } catch (IOException e) {
            Toast.makeText(getContext(), text: "Algo correu mal com a localização inserida.", Toast.LENGTH_SHORT).show();
        }
        return false;
    }
    @Override
    public boolean onQueryTextChange(String newText) {
        return false;
    }
});
```

FIGURA 60. Implementação da barra de pesquisa através de *Geocoding*.

Outro exemplo do uso desta funcionalidade é durante a criação de um marcador. Após um clique longo no mapa surge um marcador, onde as suas coordenadas são extraídas do mapa através do método *getFromLocation()* da classe *Geocoder*. Esta implementação pode ser observada na Figura 61.

```
map.setOnMapLongClickListener(new GoogleMap.OnMapLongClickListener() {
    @Override
    public void onMapLongClick(@NonNull @NotNull LatLng latLng) {
        try {
            List<Address> addresses = geocoder.getFromLocation(latLng.latitude, latLng.longitude,  maxResults: 1);
            if(addresses.size() > 0){
                Address address = addresses.get(0);
                String streetAddress = address.getAddressLine( index: 0);
                markerAdding = map.addMarker(new MarkerOptions().position(latLng).title(streetAddress).draggable(true));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
});
```

FIGURA 61. Implementação da adição de um novo marcador no mapa através de *Geocoding*.

4.9. Estrutura da Atividade *InitialPage*

Após efetuado o *Login*, o utilizador é direcionado para a Atividade *InitialPage* onde encontra uma barra de navegação no rodapé e um fragmento que apresenta o separador selecionado.

4.9.1. Barra de navegação de rodapé

A implementação da barra de navegação de rodapé implica a criação de um elemento *BottomNavigationView* no ficheiro do diretório *layout* correspondente à Atividade *InitialPage*, onde o ficheiro *bottom_nav_menu*.xml é invocado, e um ficheiro *bottom_nav_menu*.xml no diretório *menu*. A barra de navegação de rodapé implementada é composta por quatro separadores: o separador *Explorar*, o separador *Gostos*, o separador *Amigos* e o separador *Perfil*. Podemos observar a sua aparência na Figura 62.

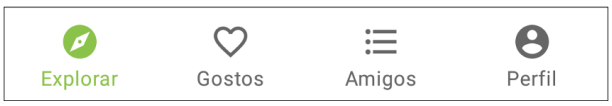


FIGURA 62. Aparência da barra de navegação de rodapé.

4.9.2. Navegação entre fragmentos

O fragmento acima da barra de navegação de rodapé é alterado consoante o separador selecionado ou a interação do utilizador no próprio fragmento. Esta alteração de fragmentos é possível através da navegação entre fragmentos implementada. Para que esta navegação fosse implementada foi necessário acrescentar dependências.

Existe um ficheiro com o nome *nav_graph.xml* no diretório *navigation* que descreve toda a navegação de fragmentos existente no projeto, ilustrada na Figura 64. Este ficheiro é invocado no elemento *fragment* no ficheiro do diretório *layout* correspondente à Atividade *InitialPage*, como observado na Figura 63.

```
<fragment
    android:id="@+id/fragmentContainerView"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_graph" />
```

FIGURA 63. Referência ao ficheiro *nav_graph.xml* no elemento *fragment*.

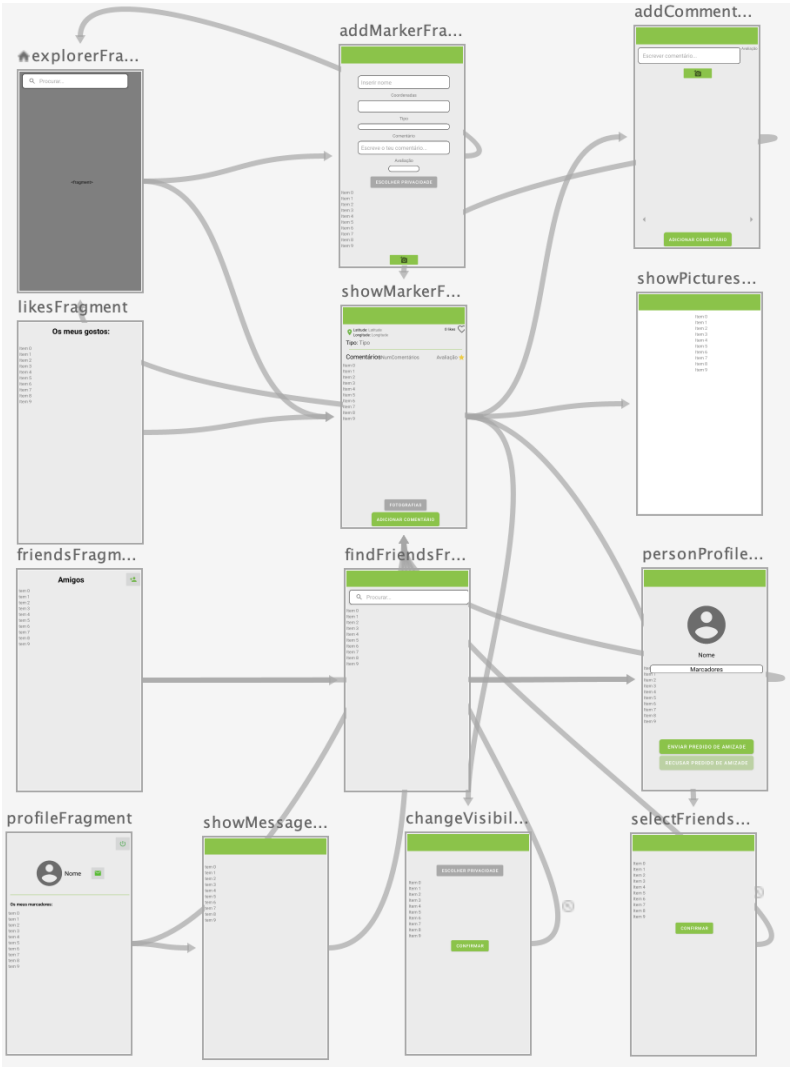


FIGURA 64. Navegação de fragmentos existente no projeto.

4.10. Localização do utilizador

Para que no mapa seja possível visualizar a localização do utilizador recorreu-se à biblioteca *FusedLocationProviderClient* da *Google* e implementar três métodos: *getLocationPermission()*, *updateLocationUI()* e *getDeviceLocation()*.

Para ter acesso à biblioteca *FusedLocationProviderClient* foi necessário declarar uma dependência.

Os três métodos referidos foram invocados no método *onMapReady()*.

No método *getLocationPermission()* é solicitada permissão ao utilizador para aceder à sua localização e, em caso afirmativo, a variável de controlo *locationPermissionGranted* passa a *true*.

No método *updateLocationUI()* é verificado se existe permissão do utilizador para aceder à sua localização através da variável *locationPermissionGranted* e, em caso afirmativo, tanto a localização do utilizador como o botão de *zoom* na localização tornam-se visíveis. Em caso negativo as duas opções são desativadas sendo invocado o método *getLocationPermission()* para que as permissões possam ser alteradas pelo utilizador, como se observa na Figura 65.

```
private void updateLocationUI() {
    if (map == null) {
        return;
    }
    try {
        if (locationPermissionGranted) {
            map.setMyLocationEnabled(true);
            map.getUiSettings().setMyLocationButtonEnabled(true);
        } else {
            map.setMyLocationEnabled(false);
            map.getUiSettings().setMyLocationButtonEnabled(false);
            lastKnownLocation = null;
            getLocationPermission();
        }
    } catch (SecurityException e)  {
        Log.e( tag: "Exception: %s", e.getMessage());
    }
}
```

FIGURA 65. Implementação do método *updateLocationUI()*.

Por fim, no método *getDeviceLocation()* é verificado se existe permissão do utilizador para aceder à sua localização através da variável *locationPermissionGranted*. Em caso afirmativo através da biblioteca *FusedLocationProviderClient* utiliza-se o método *getLastLocation* para obter a localização do utilizador que é utilizada na primeira vez em que o mapa é carregado. Se se conseguir obter a localização, esta é guardada numa variável, caso não se obtenha a localização o botão de centrar o mapa no utilizador é desativado, como se observa na Figura 66.

```
private void getDeviceLocation() {
    try {
        if (locationPermissionGranted) {
            Task<Location> locationResult = fusedLocationProviderClient.getLastLocation();
            locationResult.addOnCompleteListener(getActivity(), new OnCompleteListener<Location>() {
                @Override
                public void onComplete(@NonNull Task<Location> task) {
                    if (task.isSuccessful()) {
                        lastKnownLocation = task.getResult();
                    } else {
                        Log.d( tag: "somethingTAG",  msg: "Current location is null. Using defaults.");
                        Log.e( tag: "somethingTAG",  msg: "Exception: %s", task.getException());
                        map.getUiSettings().setMyLocationButtonEnabled(false);
                    }
                }
            });
        }
    } catch (SecurityException e)  {
        Log.e( tag: "Exception: %s", e.getMessage(), e);
    }
}
```

FIGURA 66. Implementação do método *getDeviceLocation()*.

Estes três métodos foram baseados no documento "Select Current Place and Show Details on a Map"em [**48**].

4.11. Biblioteca para apresentação de imagens

Para exibir as fotografias guardadas na base de dados nas respetivas *ImageViews*, foi decidido recorrer a uma biblioteca externa, denominada *Picasso*, por permitir o carregamento de imagens sem complicações. Esta biblioteca trata automaticamente múltiplos fatores essenciais ao processamento de imagens, tais como o redimensionamento, a *cache* e o *placeholder* enquanto a imagem é descarregada. Para tal, foi necessário recorrer a uma dependência.

De modo a apresentar as fotografias é necessário recorrer ao método *load()* da biblioteca em questão.

4.12. Remoção dos marcadores da *Google*

Sendo um dos objetivos desta aplicação a partilha de informação guardada na base de dados pelos utilizadores e, apesar de se utilizarem os mapas da *Google*, foram removidos os marcadores presentes no mapa por defeito. Apesar de em circunstâncias especiais, como na presença de um número reduzido ou até mesmo inexistente de marcadores da própria aplicação, fazer sentido manter os marcadores da *Google* de modo a complementar a informação apresentada. De modo a tornar apenas visíveis no mapa os marcadores desta aplicação foi necessário recorrer ao ficheiro *raw.json* disponibilizado pela *Google* em [**49**]. Na Figura 67 observa-se o ficheiro utilizado.

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

FIGURA 67. Ficheiro *raw.json*.

CAPíTULO 5

Resultados e Discussão

Neste capítulo são apresentados os resultados à avaliação e validação da aplicação desenvolvida através do questionário realizado, disponível no Apêndice A. O questionário é composto por dois tipos de perguntas: escolha múltipla e escala linear com os valores de 1 a 5. A escolha múltipla toma os valores de *Sim* e *Não* enquanto que a escala linear varia consoante o contexto da pergunta. Existem perguntas em que o número 1 equivale a *Mau* e o número 5 equivale a *Muito Bom*, enquanto que noutras perguntas o número 1 equivale a *Poucas Vezes* e o número 5 equivale a *Muitas Vezes*.

Foi ainda desenvolvido um manual de instruções com o objetivo de auxiliar o utilizador a melhor compreender as funcionalidades disponíveis na aplicação, disponível no Apêndice B.

5.1. Caracterização do grupo de entrevistados

A aplicação desenvolvida foi avaliada por um grupo de 10 utilizadores. Entre eles 4 do sexo feminino e 6 do sexo masculino, abrangendo uma faixa etária entre os 21 e os 25 anos.

5.2. Respostas ao questionário

Tendo em conta o conteúdo da aplicação, foi avaliada a familiaridade dos entrevistados relativamente a aplicações com conteúdo turístico e de apresentação de informação em mapa. Com as respostas apresentadas nas Figuras 68 e 69, percebemos que existe uma preferência pelas aplicações de navegação em mapa às aplicações de turismo.

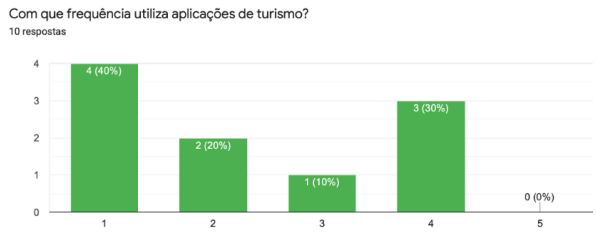


FIGURA 68. Utilização de aplicações de turismo.

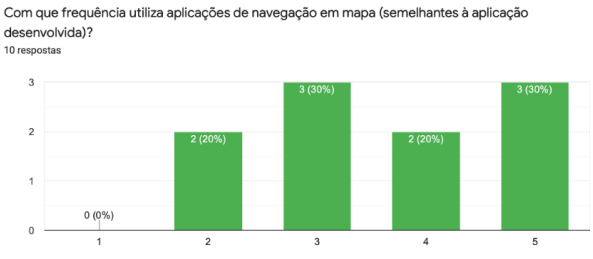


FIGURA 69. Utilização de aplicações de navegação em mapa.

No manual de instruções de utilização da aplicação estão descritas as funcionalidades possíveis. O manual considera-se bem conseguido e percetível pela maioria dos entrevistados, como de observa na Figura 70.

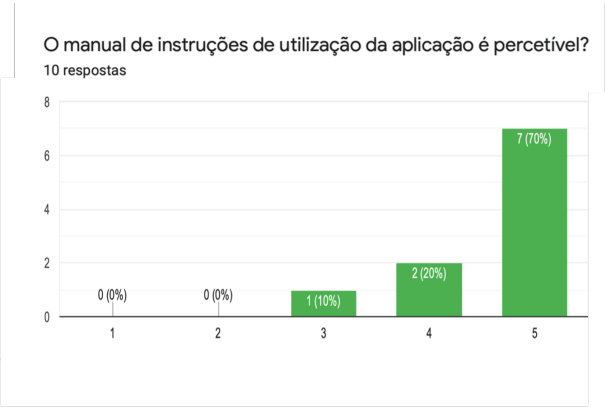


FIGURA 70. Distribuição da avaliação da perceção do manual.

FIGURA 71. Distribuição da realiação das tarefas do manual de utilização.

Pelo que se observa na Figura 71, a grande maioria dos entrevistados conseguiu realizar todas as funcionaidades. O motivo que levou a um único utilizador não realizar todas as funcionalidades foi a inexistência de amigos. Para se tirar o maior proveito do uso desta aplicação é necessário adicionar amigos, apesar de não ser um fator de critério obrigatório à utilização da aplicação.

Em relação às categorias possíveis a atribuir aos marcadores criados, 20% dos entrevistados considera que se podem adicionar novas categorias. As categorias que sugerem acrescentar são apresentadas na Figura 72.

FIGURA 72. Categorias de marcadores sugeridas a adicionar.

Também foi avaliada a privacidade dos dados sentida pelos entrevistados. Pelo que 80% considera ter a privacidade necessária, ao contrário dos restantes 20% que considera ainda existirem melhorias a fazer. Essas melhorias são enumeradas na Figura 73.

FIGURA 73. Avaliação da privacidade na aplicação.

Relativamente às funcionalidades disponíveis na aplicação, apenas 50% dos entrevistados concordaram que não era necessário acrescentar novas funcionalidades. Este foi o aspeto de maior discrepância de resultados, como observado na Figura 74.

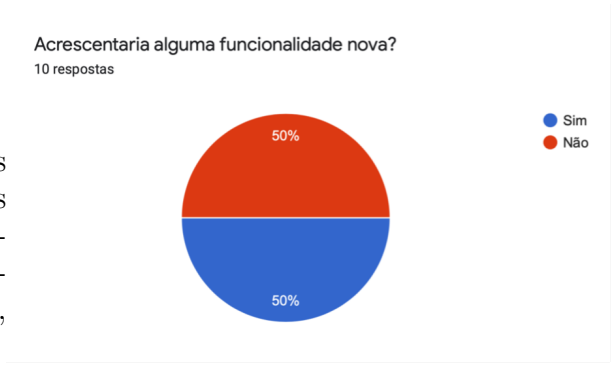


FIGURA 74. Distribuição da avaliação da necessidade de funcionalidades.

As sugestões de novas funcionalidades são apresentadas na Figura 75. Funcionalidades estas que poderão servir de proposta a um trabalho futuro.

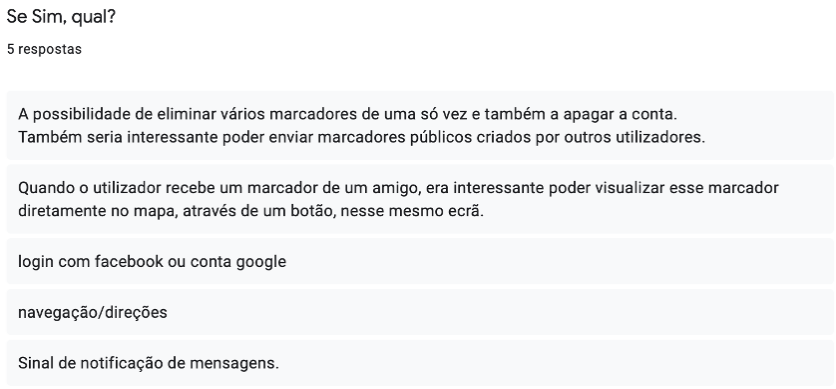


FIGURA 75. Sugestão de novas funcionalidades.

Através da avaliação apresentada na Figura 76, observa-se que a aplicação se torna útil no quotidiano do utilizador quando se efetua uma pesquisa relativa a categorias de marcadores específicas como por exemplo restaurantes. Alguns comentários referem-se ao seu uso exclusivamente em situações de viagens, como observado na Figura 77.

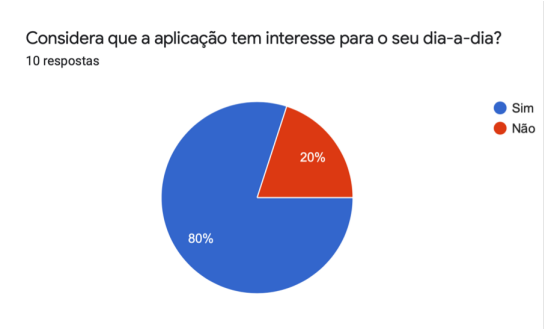


FIGURA 76. Distribuição da avaliação do uso quotidiano da aplicação pelo utilizador.

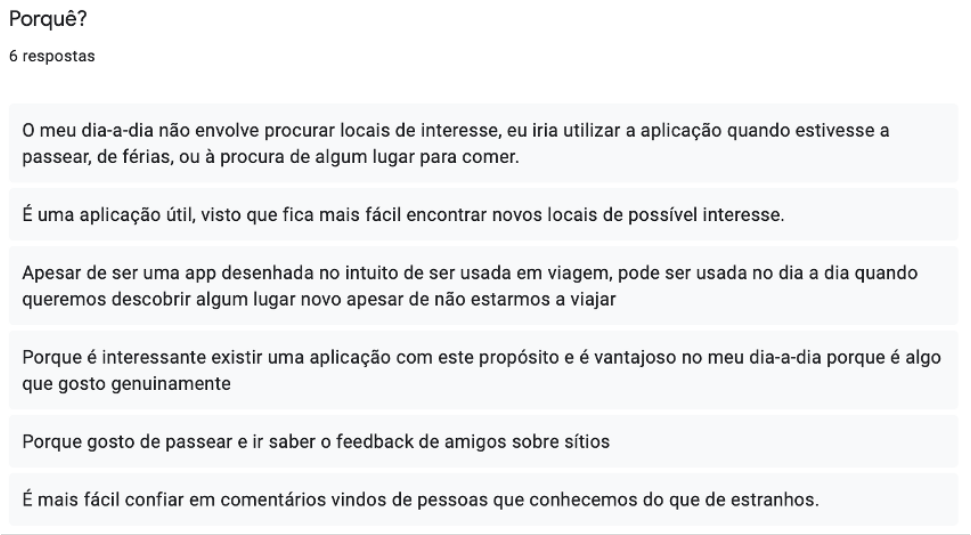
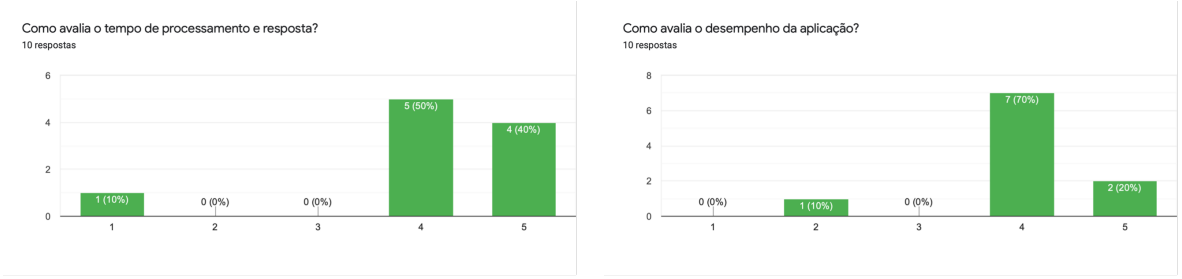


FIGURA 77. Comentários relativos ao uso quotidiano da aplicação.

Em relação ao tempo de processamento e resposta e ao desempenho, as classificações foram na sua maioria consideradas *Bom* e *Muito Bom* (classificação 4 e 5), apesar de em cada uma delas se encontrar uma classificação negativa, como se observa na Figura 78. Por vezes os tempos de resposta são maiores devidos aos pedidos feitos à base de dados que podem ser atrasados devido à ligação à *internet* ou a outros fatores externos.



(a) (b)

FIGURA 78. Distribuição da avaliação do: (a) tempo de processamento e resposta; (b) desempenho.

A *interface* da aplicação foi avaliada em vários fatores: navegabilidade no mapa intuitiva, facilidade de utilização, detalhe necessário da informação, fácil interpretação e cores utilizadas. A classificação por parte dos entrevistados foi, na maioria dos casos, considerada *Bom* e *Muito Bom* (classificação 4 e 5), como se observa na Figura 79.

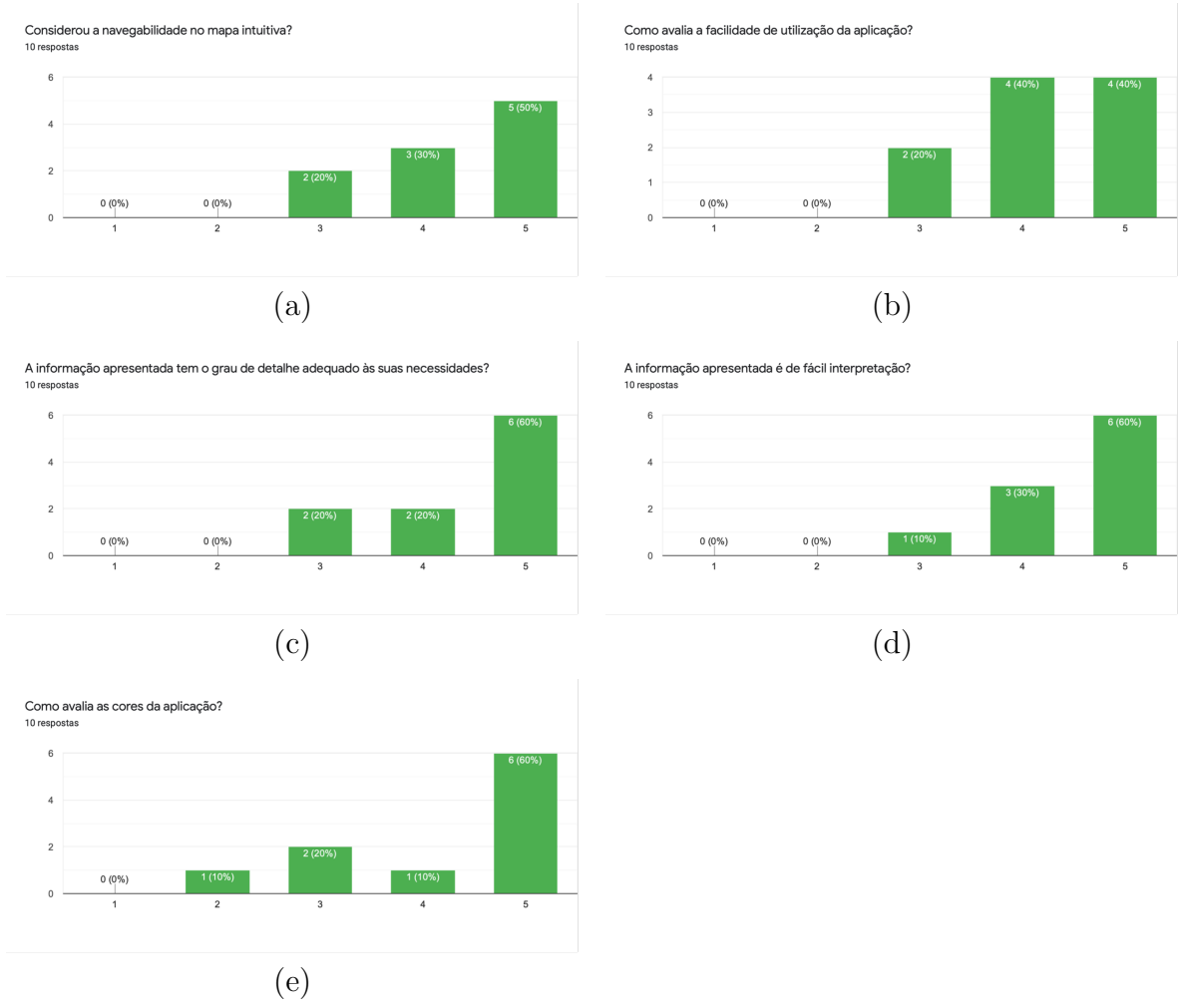
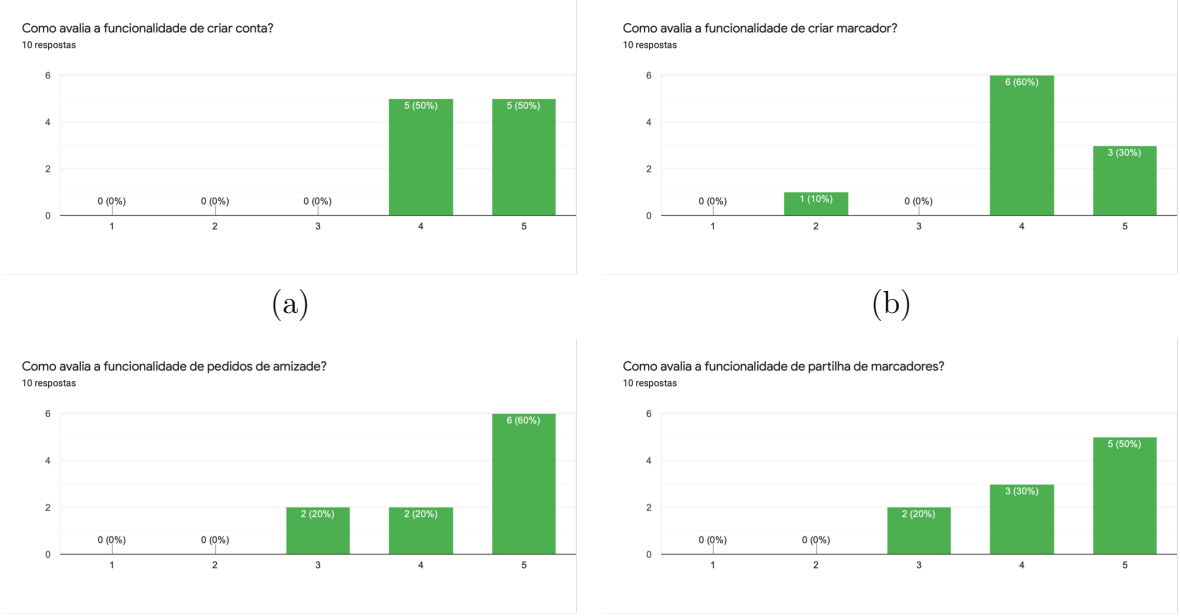


FIGURA 79. Distribuição da avaliação da aplicação: (a) navegabilidade no mapa intuitiva; (b) facilidade de utilização; (c) detalhe necessário da informação; (d) fácil interpretação; (e) cores utilizadas.

Também foi avaliada a implementação das funcionalidades de: criar conta, criar marcador, pedidos de amizade e partilha de marcadores. A classificação por parte dos entrevistados foi, na maioria dos casos, considerada *Bom* e *Muito Bom* (classificação 4 e 5). Apenas a funcionalidade de criar marcador teve uma classificação negativa em dez avaliações, como se observa na Figura 80.

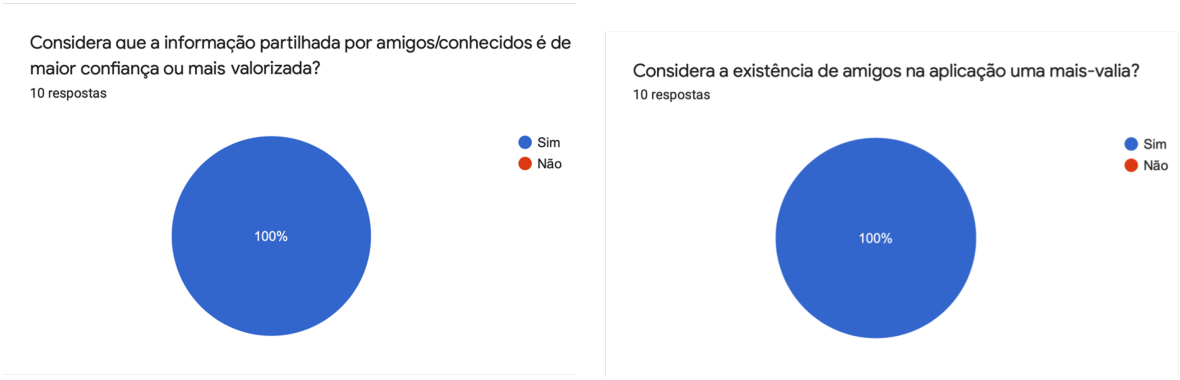


(a) (b)

(c) (d)

FIGURA 80. Distribuição da avaliação da funcionalidade de: (a) criar conta; (b) criar marcador; (c) pedidos de amizade; (d) partilha de marcadores.

A implementação de amigos na aplicação foi uma inovação e um aspeto principal desta avaliação, de modo a descobrir se realmente a informação partilhada por pessoas conhecidas se torna mais valorizada. Favoravelmente obteve-se uma avaliação positiva e unânime, onde todos os entrevistados avaliaram com nota 5 a relevância da existência de amigos, bem como a valorização da informação que se partilha entre eles, como observado na Figura 81.



(a) (b)

FIGURA 81. Distribuição da avaliação da: (a) confiança na informação partilhada; (b) valorização da existência de amigos.

No fim do questionário foi ainda aferida a satisfação geral dos entrevistados com a aplicação desenvolvida. Como se observa na Figura 82, 70% dos entrevistados ficaram bastante satisfeitos, o que significa que a aplicação correspondeu aos objetivos propostos.

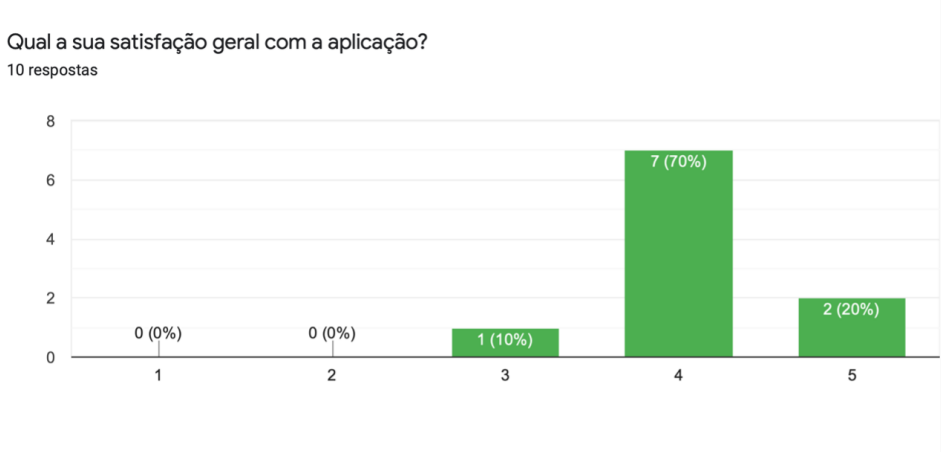


FIGURA 82. Distribuição da satisfação geral dos entrevistados.

Na Figura 83 são apresentados comentários e sugestões feitos à aplicação desenvolvida.

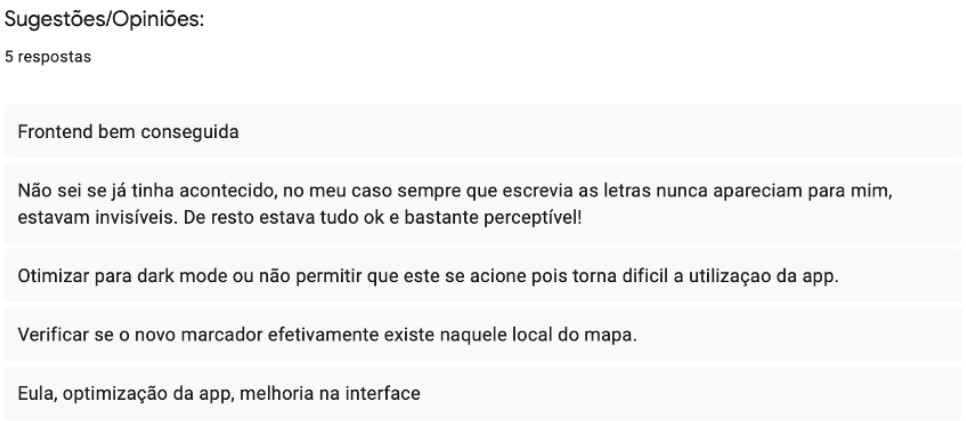


FIGURA 83. Comentários e sugestões feitos à aplicação pelos entrevistados.

CAPíTULO 6

Conclusão

Este projeto tinha como principal objetivo a criação de uma aplicação com a implementação de um SIG, com acesso e manipulação distribuídos, para *smartphones*. Foi realizada a pesquisa necessária à implementação da mesma, bem como uma avaliação por parte de um conjunto de utilizadores.

Através do questionário realizado percebeu-se que foi bem recebida pelo grupo de entrevistados, sendo uma aplicação com potencial. A apresentação de mapas com a informação presente na base de dados é de fácil interpretação e a sua navegabilidade foi considerada intuitiva, o que proporciona uma *interface* amigável e ajuda o utilizador na sua utilização. Também as funcionalidades oferecidas foram bem conseguidas e com interesse tanto para um uso quotidiano, como para um uso esporádico.

Um aspeto bastante valorizado no projeto era o impacto da interação entre utilizadores. A implementação da funcionalidade que permite a existência de amigos visava avaliar a confiança conferida pela informação partilhada por parte dos utilizadores. Através do questionário realizado verificou-se que a informação partilhada por parte de pessoas conhecidas é mais valorizada e vista como uma mais-valia.

A aplicação foi desenvolvida para *Android* e foi utilizado o *Firebase* como base de dados. Sendo uma base de dados flexível, moderna e em tempo real, permite uma menor latência no acesso aos dados. De modo a organizar o dados geográficos de forma eficiente e a aproveitar as funcionalidades do *Firebase* foi utilizado o *Firestore Database*. Permite armazenar dados do tipo *geopoint* com coordenadad retiradas diretamente do mapa, não existindo a necessidade de armazenar os dados em formato de *string*.

Resumidamente, através da aplicaçação o utilizador fornece informação que é enviada para a base de dados em tempo real. Posteriormente, a informação é dispoibilizada pela aplicação para outros utilizadores, onde é possível a sua consulta.

6.1. Limitações

Em relação ao grupo de indivíduos entrevistados, poderiam ter sido abordados um maior número de pessoas e ter sido alcançada uma faixa etária mais vasta. Proporcionado assim, avaliações com diferentes pontos de vista e com mais informação. Outro fator que limitou o grupo de inquiridos foi o facto de apenas ser possível utilizar a aplicação em *smartphones Android*.

Em relação à visualização dos marcadores no mapa pode, por vezes, ocorrer a sobreposição de marcadores devido à ampliação do mapa. A implementação de uma funcionalidade de agrupamento de marcadores pode surgir ao nível de uma proposta futura.

A apresentação de imagens na aplicação por vezes encontra-se demorada devido à comunicação com a base de dados. A qualidade da ligação à *internet* pode ser um fator determinante para o sucedido.

6.2. Trabalho Futuro

Apesar da aplicação desenvolvida ter sido bem conseguida, podem sempre ser realizadas melhorias e implementadas novas funcionalidades.

O sistema pode ser melhorado relativamente ao processamento de dados, que em vez de processados na aplicação podem ser processados ao nível do *Firebase*, o que evita a utilização de recursos por parte do telemóvel. O *Firebase* tem uma funcionalidade denominada *Firebase Cloud Functions* que permite este processo, através da escrita de funções ao nível da base de dados. Um exemplo seria a integração com a API da Google para utilização dos mapas, aumentando a velocidade de processamento cada vez que se aceder aos mapas [**50**].

Relativamente à criação de conta por parte dos utilizadores, não se realiza a verificação do *email* adicionado. Uma proposta de implementação futura seria esta mesma verificação, que iria conceder mais credibilidade e segurança à aplicação.

Todas as funcionalidades propostas pelos entrevistados no questionário realizado podem no futuro ser avaliadas e acrescentadas à aplicação.

6.3. Apreciação Final

O tema foi bastante interessante e desafiador tendo em conta que foi a primeira vez que trabalhei com dados geográficos. A realização deste projeto superou as minhas expectativas e contribuiu para o aprofundamento dos meus conhecimentos. Todas as tarefas propostas foram efetuadas tendo a dissertação sido um sucesso.

Referências Bibliográficas

[1] J. F. N. F. N. Melo, "O papel da internet e da imagem do destino turístico no turismo de lisboa," Ph.D. dissertation, Escola Superior de Hotelaria e Turismo do Estoril, 2016.

[2] N. Molina, "Viagens ajudam a aliviar cansaço mental causado pela pandemia - viagem," Aug 2021. [Online]. Available: https://viagem.estadao.com.br/noticias/geral, viagens-ajudam-a-aliviar-cansaco-mental-causado-pela-pandemia,70003825922

[3] Statista, "Number of smartphone users worldwide from 2016 to 2021," Available at https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/(2020).

[4] ——, "Mobile operating systems' market share worldwide from january 2012 to december 2019," Available at https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/(2020).

[5] Esri, "What is gis? — geographic information system mapping technology," Available at https://www.esri.com/en-us/what-is-gis/overview.

[6] G. Philipe, "Como funciona o gps?" Available at https://www.oficinadanet.com.br/post/12406-como-funciona-o-gps(2016).

[7] Wikipedia, "Sistema de posicionamento global," Available at https://pt.wikipedia.org/wiki/Sistema_de_posicionamento_global(2020).

[8] ——, "Wi-fi positioning system," Available at https://en.wikipedia.org/wiki/Wi-Fi_positioning_system(2021).

[9] R. Ariyanto, Y. W. Syaifudin, D. Puspitasari, A. Y. Ananta, A. Setiawan, E. Rohadi *et al.*, "A web and mobile gis for identifying areas within the radius affected by natural disasters based on openstreetmap data." *International Journal of Online & Biomedical Engineering*, vol. 15, no. 15, 2019.

[10] P. Doshi, P. Jain, and A. Shakwala, "Location based services and integration of google maps in android," *International Journal of Engineering and Computer Science*, vol. 3, no. 3, pp. 5072–5077, 2014.

[11] D. M. Simões and J. C. Fernandes, "Navegação indoor baseada na rede wifi como suporte a serviços baseados na localização: Estudo de caso no campus da ul," in *Atas das I Jornadas Lusófonas de Ciências e Tecnologias de Informação Geográfica.* Coimbra: Imprensa da Universidade de Coimbra, 2015, pp. 403–416.

[12] A. Parihar, "Five of the most popular databases for mobile apps," Available at https://blog.trigent.com/five-of-the-most-popular-databases-for-mobile-apps/(2017).

[13] Wikipedia, "Spatial database," Available at https://en.wikipedia.org/wiki/Spatial_database#cite_note-1(2020).

[14] ——, "Mongodb," Available at https://en.wikipedia.org/wiki/MongoDB(2021).

[15] MongoDB, "Geospatial queries — mongodb manual," Available at https://docs.mongodb.com/manual/geospatial-queries/(2021).

[16] Couchbase, "Couchbase lite," Available at https://www.couchbase.com/products/lite(2021).

[17] V. Mische, "The state of geocouch," Available at https://vmx.cx/blog/2011-09-20/the-state-of-geocouch.html(2011).

[18] SQLite, "Full-featured sql," Available at https://www.sqlite.org/fullsql.html.

[19] Wikipedia, "Spatialite," Available at https://en.wikipedia.org/wiki/SpatiaLite(2020).
[20] PostgreSQL, "Postgresql: About," Available at https://www.postgresql.org/about/(2020).
[21] PostGIS, "Postgis — postgis feature list," Available at https://postgis.net/features/.
[22] A. C. Dias and O. Postolache, "Cyclist performance assessment based on wsn and cloud technologies," in *2018 International Conference and Exposition on Electrical And Power Engineering (EPE)*. IEEE, 2018, pp. 1041–1046.
[23] Firebase, "Choose a database: Cloud firestore or realtime database — firebase," 2021. [Online]. Available: https://firebase.google.com/docs/database/rtdb-vs-firestore
[24] ——, "Tipos de dados compatíveis," Available at https://firebase.google.com/docs/firestore/manage-data/data-types(2021).
[25] Openbase, "55 best javascript map libraries," Available at https://openbase.io/categories/js/best-javascript-map-libraries?orderBy=RECOMMENDED&(2021).
[26] Mapbox, "Reference — vector tiles," Available at https://docs.mapbox.com/vector-tiles/reference/.
[27] ——, "Api reference — mapbox gl js," Available at https://docs.mapbox.com/mapbox-gl-js/api/.
[28] OpenLayers, "Openlayers - welcome," Available at https://openlayers.org/.
[29] V. Agafonkin, "Leaflet — an open-source javascript library for interactive maps," Available at https://leafletjs.com/(2020).
[30] OpenStreetMap, "Openstreetmap," Available at https://www.openstreetmap.org/about.
[31] Google, "Visão geral — maps sdk for android — google developers," Available at https://developers.google.com/maps/documentation/android-sdk/overview?hl=pt-br.
[32] ——, "What is arcgis server? - arcgis server — documentation for arcgis enterprise," Available at https://enterprise.arcgis.com/en/server/latest/get-started/windows/what-is-arcgis-for-server-.htm(2020).
[33] LikeALocal, "About us," Available at https://www.likealocalguide.com/pages/about/(2021).
[34] VisitACity, "Visit a city — about us," Available at https://www.visitacity.com/about(2021).
[35] Roadtrippers, "About — roadtrippers," Available at https://roadtrippers.com/about/(2021).
[36] Wikipedia, "Histórico de versões do android," 2021. [Online]. Available: https://pt.wikipedia.org/wiki/Histrico_de_versμes_do_Android#Android_12_(API_31)
[37] Firebase, "Firebase authentication," 2020. [Online]. Available: https://firebase.google.com/docs/auth
[38] ——, "Cloud storage for firebase," 2021. [Online]. Available: https://firebase.google.com/docs/storage
[39] ——, "Acessar dados off-line — firebase documentation," 2021. [Online]. Available: https://firebase.google.com/docs/firestore/manage-data/enable-offline?hl=pt-br
[40] ——, "Como ativar recursos off-line no android — firebase realtime database," 2021. [Online]. Available: https://firebase.google.com/docs/database/android/offline-capabilities?hl=pt
[41] F. Cordeiro, "Activity: O que é e como usar corretamente," May 2018. [Online]. Available: https://www.androidpro.com.br/blog/desenvolvimento-android/activity-intro/
[42] Android, "Fragmentos : Desenvolvedores android : Android developers," 2019. [Online]. Available: https://developer.android.com/guide/components/fragments?hl=pt-br
[43] ——, "App manifest overview : Android developers," 2021. [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro
[44] PubNub, "Defined: What is geohashing?" Mar 2021. [Online]. Available: https://www.pubnub.com/learn/glossary/what-is-geohashing/
[45] Firebase, "Geofire for android - realtime location queries with firebase," 2021. [Online]. Available: https://firebaseopensource.com/projects/firebase/geofire-android/

[46] ——, “Geo queries — firebase documentation,” 2021. [Online]. Available: https://firebase.google.com/docs/firestore/solutions/geoqueries

[47] Android, “Geocoder : Android developers,” 2021. [Online]. Available: https://developer.android.com/reference/android/location/Geocoder

[48] Google, “Selecionar o lugar atual e exibir detalhes em um mapa,” 2021. [Online]. Available: https://developers.google.com/maps/documentation/android-sdk/current-place-tutorial

[49] ——, “Como ocultar elementos do mapa com estilos,” 2021. [Online]. Available: https://developers.google.com/maps/documentation/android-sdk/hiding-features

[50] Firebase, “What can i do with cloud functions? firebase documentation,” 2021. [Online]. Available: https://firebase.google.com/docs/functions/use-cases

APÊNDICE A

Questionário

Idade *

A sua resposta

Sexo (F/M) *

A sua resposta

1. Com que frequência utiliza aplicações de turismo? *

	1	2	3	4	5	
Poucas vezes	○	○	○	○	○	Muitas vezes

2. Com que frequência utiliza aplicações de navegação em mapa (semelhantes à aplicação desenvolvida)? *

	1	2	3	4	5	
Poucas vezes	○	○	○	○	○	Muitas vezes

3. Conseguiu realizar todas as tarefas presentes no manual de instruções? *

- ○ Sim
- ○ Não

Se Não, quais?

Texto de resposta longa

4. Considera que faltam categorias de marcadores? *

- ○ Sim
- ○ Não

Se Sim, quais?

Texto de resposta longa

5. Encontra algo a melhorar em termos de privacidade? *

- ◯ Sim
- ◯ Não

Se Sim, o quê?

Texto de resposta longa

6. Acrescentaria alguma funcionalidade nova? *

- ◯ Sim
- ◯ Não

Se Sim, qual?

Texto de resposta longa

7. Considera que a aplicação tem interesse para o seu dia-a-dia? *

- ◯ Sim
- ◯ Não

Porquê?

Texto de resposta longa

8. O manual de instruções de utilização da aplicação é percetível? *

	1	2	3	4	5	
Mau	◯	◯	◯	◯	◯	Muito Bom

9. Considerou a navegabilidade no mapa intuitiva? *

	1	2	3	4	5	
Mau	◯	◯	◯	◯	◯	Muito Bom

10. Como avalia a facilidade de utilização da aplicação? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

11. A informação apresentada tem o grau de detalhe adequado às suas necessidades? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

12. A informação apresentada é de fácil interpretação? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

13. Como avalia as cores da aplicação? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

14. Como avalia a funcionalidade de criar conta? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

...

15. Como avalia a funcionalidade de criar marcador? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

...

16. Como avalia a funcionalidade de pedidos de amizade? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

17. Como avalia a funcionalidade de partilha de marcadores? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

18. Como avalia o tempo de processamento e resposta? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

19. Como avalia o desempenho da aplicação? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

20. Qual a sua satisfação geral com a aplicação? *

	1	2	3	4	5	
Mau	○	○	○	○	○	Muito Bom

21. Considera que a informação partilhada por amigos/conhecidos é de maior confiança ou mais valorizada? *

- ○ Sim
- ○ Não

22. Considera a existência de amigos na aplicação uma mais-valia? *

- ○ Sim
- ○ Não

Sugestões/Opiniões:

Texto de resposta longa

Muito obrigada pelo tempo dispensado!

APÊNDICE B

Manual de Instruções de Utilização

Este manual é composto pelas instruções necessárias à instalação e utilização da aplicação. O manual baseia-se nas seguintes secções:

- Instalar aplicação;
- Criar conta;
- Adicionar marcador ao mapa;
- Visualizar marcador do mapa;
- Enviar marcador;
- Visualizar marcador recebido;
- Adicionar comentário a marcador;
- Adicionar um amigo;
- Visualizar marcador de um amigo.

Instalar aplicação:

1. Descarregar e instalar o ficheiro *apk* apresentado na Figura 1;

Figura 1 - Ficheiro apk para instalação da aplicação no telemóvel.

Criar conta:

1. Preencher os campos apresentados e premir o botão *Criar Conta*, como ilustrado na Figura 2;

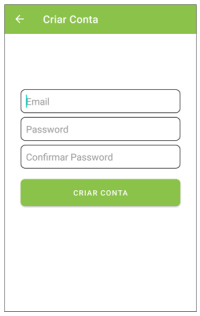


Figura 2 - Campos a preencher na criação de conta.

2. Surge um segundo ecrã, ilustrado na Figura 3. Preencher e premir o botão *Guardar* para finalizar a criação de conta.

Figura 3 - Segundo ecrã apresentado para a criação de conta.

Adicionar marcador ao mapa:

1. Fazer um clique longo no ecrã até aparecer um ícone de localização vermelho (fazendo um clique longo sobre o ícone vermelho pose reposicionar o marcador);
2. Carregar sobre esse ícone e de seguida sobre a janela de informações desse mesmo ícone, como apresentado na Figura 4;

Figura 4 - Mapa com ícone de localização vermelho e respetiva janela de informações.

3. Preencher os campos apresentados na Figura 5.

Figura 5 - Campos a preencher para a criação de um marcador novo.

4. Premir o botão com símbolo de câmara fotográfica para selecionar uma ou mais fotografias e premir o botão *Adicionar* para finalizar a criação de um marcador, como apresentado na Figura 6.

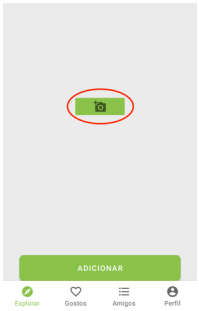


Figura 6 - Adicionar fotografia/as ao marcador.

Visualizar marcador do mapa:

1. Carregar sobre um marcador no mapa e de seguida sobre a janela de informações respetiva. As informações do marcador aparecem como ilustrado na Figura 7.

Figura 7 - Apresentação do marcador.

Enviar marcador:

1. Visualizar marcador;
2. Carregar no botão com o ícone de seta na barra de topo, ilustrado na Figura 8;

Figura 8 - Barra de topo.

3. Selecionar na lista os amigos a quem se pretende enviar o marcador, os amigos selecionados aparecem a verde-claro, e carregar no botão *Confirmar*, como ilustrado na Figura 9.

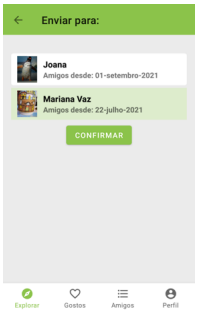


Figura 9 - Lista de amigos a selecionar.

Visualizar marcador recebido:

1. Abrir o separador do *Perfil* e selecionar o botão com ícone de uma carta, ilustrado na Figura 10;

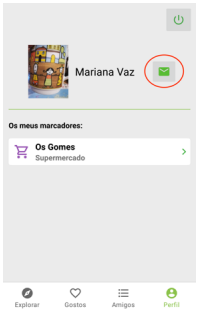


Figura 10 - Ícone a premir para visualizar marcadores recebidos.

2. Os marcadores recebidos são apresentados em lista, como ilustrado na Figura 11.

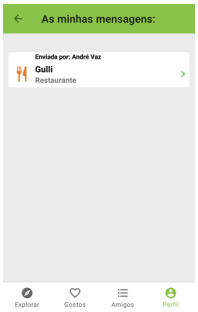


Figura 11 - Apresentação de marcadores recebidos.

Adicionar comentário a marcador:

1. Carregar sobre um marcador no mapa e de seguida sobre a janela de informações respetiva;
2. Preencher os campos apresentados e premir o botão com ícone de câmara fotográfica para adicionar fotografias (procedimento explicado na secção *Adicionar marcador ao mapa* ponto 4.), como ilustrado na Figura 12;
3. Premir o botão *Adicionar Comentário* para finalizar a adição de um novo comentário, como ilustrado na Figura 12.

Figura 12 - Campos a preencher para a adição de um novo comentário.

Adicionar um amigo:

1. Selecionar o separador *Amigos* na barra de navegação de rodapé.
2. Carregar no botão com o ícone +, como ilustrado na Figura 13;

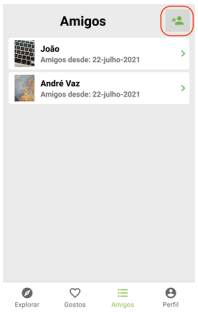


Figura 13 - Ícone a premir para adicionar um novo amigo.

3. Pesquisar pelo nome do amigo pretendido na barra de pesquisa caso seja necessário, como ilustrado na Figura 14;

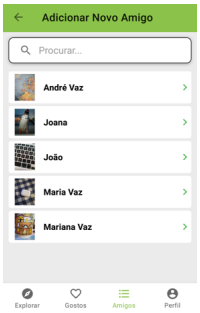


Figura 14 - Procurar amigo.

4. Selecionar o amigo e carregar no botão *Enviar Pedido de Amizade*, como ilustrado na Figura 15.

Figura 15 - Enviar Pedido de Amizade.

Visualizar marcador de um amigo:

1. Selecionar o separador *Amigos* na barra de navegação de rodapé;
2. Selecionar um amigo;

3. Selecionar um marcador presente na lista de marcadores apresentada no perfil do amigo, como ilustrado na Figura 16.

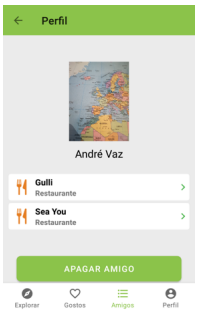


Figura 16 - Perfil do amigo com os respetivos marcadores.