Seguimento e registo de viagens por smartphone

Tiago Mário Morgado Mendes

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Frutuoso Silva

27 de Fevereiro de 2021

Agradecimentos

Tenho primeiro de agradecer, à minha família pela ajuda, principalmente à minha mãe que sempre me incentivou a continuar com a dissertação e a finalizar a mesma. Agradecer ao Professor Doutor Frutuoso Silva, que foi o meu orientador, por ter tanta paciência, por me orientar no caminho certo e pela disponibilidade.

Gostaria de agradecer aos meus amigos que sempre, de alguma forma me incentivaram e encorajaram a alcançar esta meta: André Monteiro, Luís Rodrigues, Diogo Monteiro, Tiago Raposo, Pedro Monteiro, Ianca Feitosa e João Saraiva.

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

IDE Ambiente de desenvolvimento integrado

API Application Programming Interface

XML Extensible Markup Language

JSON JavaScript Object Notation

GPS Sistema de Posicionamento Global

W3C World Wide Web Consortium

SO Sistema Operativo

BaaS Backeend-as-a-Service

PHP Hypertext Preprocessor

URL Uniform Resource Locator

UI Interface do utilizador

Capítulo 1

Introdução

Este documento foi elaborado no âmbito da Unidade Curricular de Dissertação, inserida no Mestrado em Engenharia Informática da Universidade da Beira Interior e apresenta todo o trabalho desenvolvido durante o desenrolar da mesma. Inicialmente será contextualizado o projeto, a área em que se enquadra, seguido da motivação, objetivos e, por fim, a organização do documento.

1.1 Enquadramento e Motivação

O número de Smartphones tem vindo a aumentar, crescendo cerca de 10% entre o ano de 2019 e o de 2020, tendo sido o número de Smartphones reportado no ano de 2019 de 3.2 Biliões e no ano de 2020 o número reportado foi de 3.5 Biliões, este dado foi retirado com base na *cuponation*[1] e na *statista*[2]. Com o aparecimento do Sistema de Posicionamento Global (GPS) e principalmente com a implementação deste sistema nos nossos dispositivos móveis começaram a ser desenvolvidas aplicações de rastreamento.

Este tipo de aplicações foi usado com um variado leque de funcionalidades. Como por exemplo controlo parental pois aplicações de rastreamento são uma boa opção para monitorizar o percurso de uma determinada pessoa, neste caso familiares.

A maioria das aplicações de rastreamento usadas em 2020 foi com o objectivo de rastrear as encomendas ao domicilio, isto deveu-se ao aumento das entregas de comida ao domicilio, com o aparecimento das restrições de isolamento.

Na nossa aplicação o *GPS* foi utilizado com o intuito de melhorar a organização de

um determinado evento e com isto saber as localizações de todos os participantes do Evento, tal como analisar o percurso percorrido por um utilizador, durante um evento.

1.2 Objetivos

Este projeto tem como objetivo a construção de uma aplicação móvel, com um variado leque de funções, de entre as quais se realçam as seguintes:

- Criação de Eventos;
- Possibilidade de os utilizadores de um evento comunicarem entre si, através de um Chat simples;
- Dispor as posições atuais de todos os utilizadores de um evento em um mapa;
- Ver o trajeto percorrido por um utilizador;
- Ser possível guardar as imagens tiradas por um utilizador e visualizar as mesmas;

1.3 Abordagem

Em primeiro lugar, foram estudadas outras aplicações semelhantes. Em segundo foram estudadas todas as ferramentas que seriam necessárias para conseguir concretizar todos os objetivos deste projeto, como qual servidor utilizar para guardar toda a informação fundamental para que a aplicação cumprisse com os seus objetivos pretendidos. Que linguagem e ambiente utilizar para construir o Front-end desta aplicação. Em terceiro lugar comecei a implementar a aplicação tendo começado pela questão de criar eventos, seguido pela criação do Chat. Em último lugar implementei o mapa onde permite visualizar as posições de todos os utilizadores do evento, bem como a possibilidade de adicionar imagens e a visualização das mesmas.

1.4 Organização do Documento

De modo a refletir o trabalho que foi realizado, foi entendido estruturar o relatório da seguinte forma:

- o primeiro capítulo – **Introdução** – apresenta o projeto realizado, com uma breve explicação das razões que levaram à escolha do mesmo, o objetivo pretendido, assim como a forma de organização do documento.

- o segundo capítulo – **Estado da Arte / Trabalhos relacionados** – Descreve algumas aplicações semelhantes com a aplicação a desenvolver.

- o terceiro capítulo – **Engenharia de Software** – descreve os requisitos funcionais e não funcionais. É apresentado também um Diagrama de casos de uso e um Diagrama de classes, assim como comentários relativos ás coleções de dados usadas na base de dados.

- o quarto capítulo – **Tecnologias e Ferramentas** – apresenta todas as ferramentas utilizadas no desenvolvimento do projeto, as bases de dados, o Ambiente de desenvolvimento integrado (IDE) utilizado, a linguagem de programação, bem como todos os métodos, bibliotecas e `framework` que foram necessários adicionar para a realização deste projeto.

- o quinto capítulo – **Desenvolvimento** – descreve os pormenores de desenvolvimento da aplicação móvel, mostrando alguns excertos de código e explicar os mesmos.

- sexto capítulo – **Manual de Utilização e Testes** – apresenta o manual de instruções da aplicação, bem como alguns testes de funcionamento e performance efetuados.

- o sétimo capítulo – **Conclusões e Trabalho Futuro** – descreve as conclusões retiradas da realização do projeto, bem como todo o trabalho a realizar no futuro de modo a complementar e expandir o estudo, e trabalho realizado.

Capítulo 2

Estado da Arte / Trabalhos Relacionados

2.1 Introdução

Neste capítulo num primeiro momento vamos referir e analisar algumas aplicações semelhantes. Em seguida referir qual a plataforma escolhida para a programação desta aplicação móvel e a razão dessa escolha, como também do Sistema Operativo (SO).

2.2 Plataformas Móveis

Este projeto foi desenvolvido sobre uma plataforma móvel, devido a ser uma aplicação de rastreamento usando o GPS, isto fez com que fosse necessário que a aplicação corresse em uma plataforma de pequeno porte (smartphones), para que o utilizador podesse levar consigo. Outro factor importante é que todos os dispositivos móveis de hoje possuem a capacidade de localização usando GPS.

O sistema operativo escolhido foi o Android visto que é o SO mais dominante no mercado, retendo 72% do mercado, principalmente na Europa (ver figuras 2.1 e 2.2), apesar de o SO dominante na América ser o iOS[3].

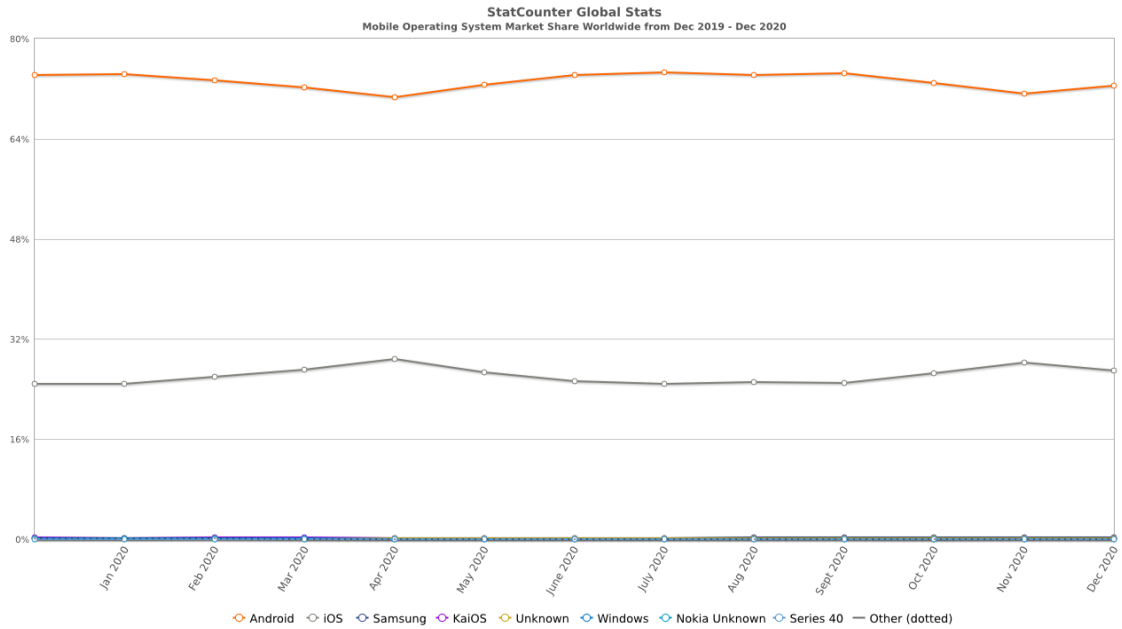

Figura 2.1: Sistemas operativos Global

Region	Android OS	iOS	KaiOS	Samsung	Unknown
North America	48.55%	51.24%	XX%	0.13%	0.02%
Asia Pacific	82.74%	16.52%	0.25%	0.23%	0.11%
Europe	73.32%	26.25%	XX%	0.24%	0.02%
South America	89.13%	10.5%	XX%	0.26%	0.02%
Africa	85.79%	12.3%	XX%	XX%	1.21%

Figura 2.2: Sistemas operativos segundo região

2.3 Aplicações similares

2.3.1 Find my friends

Find my friends[4] permite facilmente localizar amigos e família. Esta aplicação está disponível para iPhone, iPod touch ou iPad. Para visualizar a sua localização atual basta permitir partilhar a localização e esta vai ficar disponível para os seus amigos, podendo facilmente deixar de partilhar.

Esta aplicação permite também criar alertas que notificam automaticamente o utilizador em caso de algum evento, como por exemplo a chegada de seu filho à escola ou

ao aeroporto.

Principais funcionalidades:

- Permite facilmente localizar amigos e família;
- Notificações com base na localização/evento;
- Partilha de localização de forma permanente ou temporária.

2.3.2 GPS Tracking Pro

GPS Tracking Pro permite ver o seu percurso no mapa, podendo fazer pausa e retornar mais tarde. É possível também guardar o percurso e partilhar este com quem quisermos ex: amigos[5].

A aplicação permite ver várias estatísticas como:

- Velocidade máxima;
- Velocidade média;
- Duração;
- Número de posições guardadas;
- Altitude máxima;

Esta aplicação é mais para fazer o rastreamento do utilizador, permitindo depois analisar e partilhar o percurso efetuado.

2.3.3 GeoTracker

A GeoTracker é uma aplicação de rastreamento GPS para Android[6]. Tem várias funcionalidades como:

- Gravar trilhas e visualizar as mesmas usando o mapa da google ou Yandex;
- Importar e exportar trilhas;
- Visualizar várias estatísticas das trilhas como distância, velocidade máxima e média, bem como diferença de elevação e duração entre outras estatísticas;

- Marcar pontos com notas de texto. Esta aplicação tem uma classificação de 4.1, com mais de 59 mil avaliadores.

Esta aplicação permite fazer o rastreamento do utilizador para depois analisar e partilhar o percurso efetuado. Neste caso permite ainda adicionar notas ao percurso/-trilhas.

2.3.4 iSharing

A aplicação *iSharing*[7] é muito similar à aplicação *Find My friends* tendo como objetivo principal, o controlo parental, possuindo as mesmas funcionalidades referidas no *Find my Friends* com o acrescento de:

- Permite enviar um Alerta de Pânico numa situação de emergência, bastando para isso agitar/abanar o telemóvel;
- Rastreador de localização GPS para telefones roubados ou perdidos;
- Transforma o telemóvel em um walkie-talkie com *iSharing*, através das mensagens de voz gratuitas;

Como podemos ver a maioria deste tipo de aplicações ou se destina a efetuar um controlo parental ou a registar o percurso efetuado pelo utilizador para posterior análise e partilha. No caso da nossa aplicação o que se pretende é ter uma forma de seguimento de um grupo de utilizadores, onde além disso podem comunicar entre si e adicionar fotografias ao percurso, como também ver o próprio percurso e alguns dados do mesmo.

Capítulo 3

Engenharia de Software

3.1 Introdução

Neste capítulo num primeiro momento vamos referir os requisitos funcionais e requisitos não funcionais. Em seguida referir como foram organizados os dados da aplicação e por último mostrar o Diagrama de casos de uso, o Diagrama de Classes e o Diagrama de Atividades desta aplicação.

3.2 Requisitos

3.2.1 Requisitos Não Funcionais

Os requisitos não funcionais da aplicação são os seguintes:

- A Application Programming Interface (API) mínima é a 21 (Android 5.0);
- A API indicada é a 28;
- O dispositivo deverá ter ligação *Wi-Fi* ativa para se poder conectar ao servidor BackEnd;
- A aplicação deverá ser intuitiva, minimalista e de fácil utilização;

3.2.2 Requisitos Funcionais

ID	Título	Descrição	Detalhe
RF1	Login	O utilizador deverá se poder autenticar na aplicação, utilizando os dados com os quais fez o registo na mesma.	Composto pelo email e a password (inicialmente a password está escondida, podendo escolher visualizar a mesma, no caso de o utilizador assim o pretender fazer). O login é necessário para que o utilizador possa fazer qualquer operação na aplicação. O login pode ser feito também com o provedor da Google
RF2	Registar	Permitir o utilizador se registar na aplicação.	Para o utilizador se registar é necessário que preencha o username, email, password e que confirme a password. O utilizador pode também escolher uma imagem como imagem de perfil ou ficar com a imagem pré-definida pela aplicação.
RF3	Criar Evento	O utilizador poderá criar um evento para que outros utilizadores se possam juntar a este.	Para criar um evento o utilizador deverá escolher um nome, descrição, data, país, cidade e rua.
RF4	Listar eventos	Permitir ao utilizador entrar num evento, podendo ser o autor do evento ou não.	Para isso basta o utilizador escolher um dos eventos disponíveis. No ecrã de listar Eventos é possível filtrar os eventos, por criador do evento ou por nome do evento. Tem também a opção de escolher listar apenas os próprios Eventos.
RF5	Log out	Permitir ao utilizador sair da aplicação.	Ao fazer log out, o utilizador depois pode voltar a entrar com outro utilizador (podendo também entrar com o mesmo) ou fazer um novo registo.
RF6	Ecrã Principal	Ecrã onde permite ao utilizador, criar um evento, listar eventos, ver informação do utilizador ou fazer logout.	Ao fazer login ou o registo, o utilizador é direccionado para o ecrã principal onde pode escolher fazer Logout, criar um evento ou listar todos os eventos.
RF7	Ecrã de um evento	Permitir ao utilizador abrir um chat, um mapa ou sair do evento. No caso de o utilizador ser o proprietário do evento este pode fechar o Evento ou eliminar o Evento	Ao clicar em um evento, o utilizador é direccionado para o ecrã desse evento, onde pode abrir um chat, um mapa ou sair do evento.
RF8	Chat	Permite ao utilizador comunicar com os outros participantes daquele evento.	Permite trocar mensagens com os utilizadores do evento.
RF9	Mapa	Permite ao utilizador ver a sua posição no mapa, tal como os vários pontos por onde o utilizador já passou. É possível também ver as várias imagens postadas pelos participantes do evento, e a localização atual dos outros participantes do evento.	Permite ver no mapa os utilizadores do evento e o seu trajeto, bem como, poder aceder a imagens publicadas por eles.
RF9.1	Marca de participante	Marca que mostra a posição de um participante.	Ao clicar na marca que pertence ao próprio utilizador, este pode adicionar uma imagem, é possível adicionar qualquer número de imagens apesar de só poder ser adicionada uma imagem de cada vez. Podendo também visualizar informação sobre o mesmo, como a distância percorrida pelo utilizador, tal como a velocidade média a que percorreu essa distância.
RF9.2	Marca de Imagem	Marca que representa uma imagem ou várias imagens.	É possível ter várias imagens numa localização apesar de ser só possível ver uma no mapa. Ao clicar na imagem é depois possível ver as outras imagens.
RF9.3	Visualizar imagens	O utilizador pode visualizar as várias imagens que foram adicionadas pelos participantes de um Evento.	Para visualizar imagens é necessário, clicar ou numa marca de uma imagem ou numa marca de um participante na qual essa posição tem também imagens.
RF9.4	Ecrã de adicionar Imagem	Permite ao utilizador adicionar uma imagem ao evento.	Só é possível adicionar imagens a sua posição atual, para fazer isto o utilizador precisa clicar na sua posição e clicar em *adicionar imagem.* No ecrã de adicionar imagem o utilizador precisa escolher um título e uma imagem, podendo também adicionar uma descrição. Depois de adicionar a imagem não é possível modificar esta informação.
RF9.5	Percurso percorrido	O trajeto percorrido pelo utilizador é mostrado no mapa por pontos verdes ou vermelhos, dependendo do tempo demorado a percorrer o trajecto. Mostra pontos verdes se o tempo entre dois pontos for inferior a 3 minutos e vermelhos se for superior.	Apenas é possível ver o próprio percurso percorrido e não o dos outros participantes do evento.
RF9.6	Eliminar Imagem	A opção de eliminar uma imagem está disponível no ecrã de visualizar imagem.	Para eliminar uma imagem é preciso ser o criador da própria.
RF10	Fechar Evento	Fechar um evento faz com que o mesmo fique disponível só para visualização.	Ao fechar um evento não é possível adicionar imagens, nem enviar mensagens. Podendo visualizar as imagens e mensagens já existentes no Evento. A opção de fechar um evento encontra-se no ecrã principal do evento, estando apenas disponível para o autor do Evento.

3.3 Organização dos Dados

3.3.1 Base de dados Relacional

A Base de dados relacional usa tabelas para a sua organização, evitando a redundância de informação, sendo cada tabela composta por colunas e linhas. Antes de se construir uma tabela é desenhado um esquema que é a planta, na qual se constrói depois a devida tabela ou tabelas[8]. Numa base de dados relacional a informação é mais consistente devendo seguir as propriedades *ACID* (atomicidade, consistência, isolamento e durabilidade).

Atomicidade: Qualquer transação que seja composta por várias partes apenas é executada, se esta for executada na totalidade caso contrário, esta transação é revertida na totalidade.

Consistência: Uma transação cria um estado novo, guardando o estado anterior, para no caso de erro, este possa recuperar o estado anterior.

Isolamento: Uma transação ainda em curso não pode ter qualquer conflito com outra transação a correr concorrentemente.

Durabilidade: A informação de uma base de dados é guardada de tal forma que em caso de acontecer um reinício ou um falha repentina, não se perca nenhuma informação.

No entanto, para a aplicação optou-se por não usar uma base de dados relacional, mas sim uma base de dados não relacional.

3.3.2 Base de dados não relacional

Uma base de dados não relacional não precisa de um esquema rígido de tabelas dando a liberdade de guardar informação sem fazer uma profunda análise do esquema necessário para a construção da mesma. Os dados não são relacionais, ou seja, no meu caso usando uma base de dados relacional, se eu tivesse um evento para saber os utilizadores desse evento teria de ir a uma tabela buscar as relações de eventos com utilizadores e depois ir buscar a informação de cada utilizador.

Com uma Base de dados não relacional os dados estão estruturados de uma forma hierárquica, no caso da minha base de dados, os utilizadores estão associados a cada evento apesar de isto ocupar mais espaço, melhora bastante a performance de uma pesquisa como buscar todos os utilizadores de um evento.

Um exemplo ainda melhor disto seria as mensagens enviadas em determinado evento, como tenho todas as mensagens de um evento inseridas numa coleção, estando está coleção associada a um Evento, isto torna uma pesquisa para saber quais as mensagens enviadas num determinado evento bastante rápida, visto que não teria de filtrar as mensagens, estando todas as mensagens pertencentes ao Evento dentro desta coleção, na figura 3.1 podemos ver 4 documentos dentro da coleção chat, estes 4 documentos representam 4 mensagens que pertencem ao Evento "Testar Mensagens".

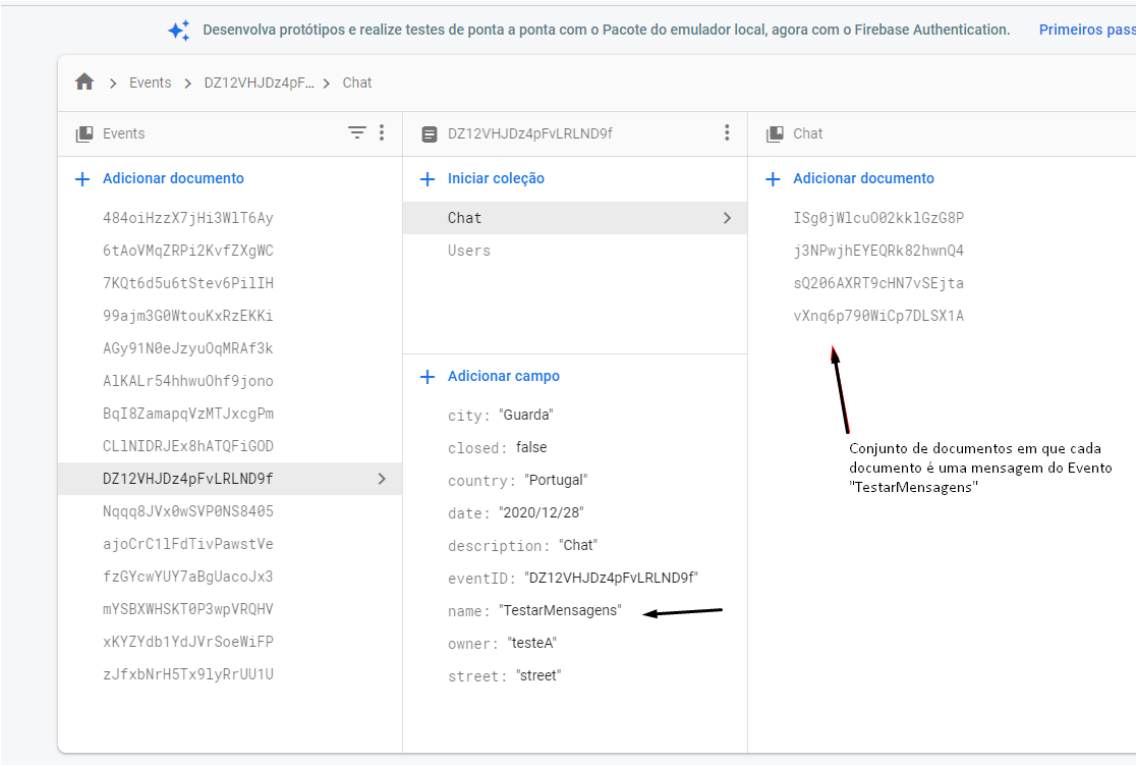

Figura 3.1: Mensagens dentro do evento "TestarMensagens"

Uma base de dados não relacional é **descentralizada** em vários provedores de base de dados, alguns casos em nuvem ou auto-hospedados. Isto aumenta a disponibilidade da base de dados visto que podemos ter a informação replicada em diferentes servidores e no caso de um servidor estar em baixo, é possível obter a informação de outro servidor.

A base de dados não relacional pode também reduzir a utilização de recursos no caso de dados soltos, ou seja, na base de dados relacional podemos ter uma linha em que várias colunas estão vazias, apesar de estas colunas estarem vazias estas vão ocupar espaço, isto numa base de dados não relacional não acontece, visto que um documento não precisa de ter uma estrutura definida.

Para aumentar a disponibilidade e latência de uma consulta, as base de dados não relacionais podem usar o serviço Inconsistência eventual. Por exemplo, um tweet feito em Portugal pode resultar numa copia escrita num servidor no Peru. Isto facilita a performance global. Isto não só tem a desvantagem de ocupar mais espaço como também, em que uma consulta realizada em simultâneo aos dois servidores, possa vir com uma inconsistência de dados, sendo depois esta inconsistência resolvida quando finalmente os dois servidores ficarem em sincronia. Para o exemplo do tweet feito em Portugal, se um utilizador fizer uma consulta no Peru ao mesmo tempo que este tweet é feito em Portugal este ainda não foi escrito na base de dados do Peru, estando as duas bases de dados nesse momento com dados inconsistentes até que esse tweet seja escrito na base de dados do Peru.

3.4 Organização dos Dados na aplicação

Como referido na secção anterior para esta aplicação foi usada uma Base de dados não relacional estando os dados organizados por coleções. Nesta secção vou detalhar cada coleção que foi utilizada neste projeto para guardar a informação necessária para o funcionamento da mesma.

3.4.1 Coleção Eventos

Constituída pelos seguintes atributos:

- City : Cidade do Evento;
- Date: Data do Evento;
- Country: País do Evento;
- Description: Descrição do Evento;
- eventID: Identificador único do Evento;
- name: Nome do Evento;
- owner: Username do criador do Evento;
- street: Rua do Evento;

- closed: Indica se um Evento está fechado ou não;

Existem quatro coleções associadas a coleção Eventos:

- Coleção de Users - coleção composta por todos os utilizadores pertencentes ao Evento;
- Coleção ImageMarkers - coleção composta por todas as Imagens pertencentes ao Evento;
- Coleção Chat - coleção composta por todas as mensagens do Evento;
- Coleção UserPositions - coleção composta por todas as posições de um utilizador em determinado Evento, esta coleção está contida na coleção Users;

3.4.2 Coleção de Users

- Email: Endereço de um email de um dado utilizador;
- mImageUrl: Url da imagem de um utilizador;
- user_id: identificador único de um utilizador;
- username: Nome de um determinado utilizador;
- Password: Password de um determinado utilizador;

3.4.3 Coleção ImageMarkers

Esta coleção tem os seguintes atributos:

- Description: Descrição de uma imagem (campo não obrigatório) ;
- eventId: Identificador único do evento a que pertence a imagem;
- geoPoint: localização geográfica (latitudo e longitude) da imagem;
- imageName: Nome da imagem;
- imageUrl: Url da imagem (apontando para o firebaseStorage onde se encontram as imagens);

- timestamp: Data em formato string (dd M aaaa hh:mm:ss);
- user_id: identificador único do utilizador que criou a imagem;

3.4.4 Coleção Chat

Esta coleção tem os seguintes atributos:

- eventId: Número único que identifica um Evento;
- messageBody: Conteúdo da mensagem;
- sender: identificador único do utilizador que enviou a mensagem(username);
- time: data em formato de número, este campo serve para ordenar as mensagens de forma cronológica;

3.4.5 Coleção UserPositions

Esta coleção tem os seguintes atributos:

- distanceTraveled: distância total percorrida por um dado utilizador num evento em metros;
- velocity: velocidade média com a qual o utilizador percorreu o seu trecho em km/h;
- lastPosition: coordenadas geográficas da ultima posição do utilizador;
- Coleção Positions: coleção que guarda todas as posições do utilizador em determinado Evento;

3.4.6 Coleção Positions

Esta coleção tem os seguintes atributos:

- time: Data em que o utilizador passou em determinada localização (representada em formato número em milisegundos);
- geoPoint: posição geográfica de um determinado utilizador;

3.4.7 Coleção User Location

Esta coleção tem os seguintes atributos:

- User: Classe User (todos os atributos da Classe User)

- geoPoint: Posição geográfica de um determinado utilizador;

- timeStamp: Data em que o utilizador passou em determinada localização (representada em formato número em milisegundos);

3.5 Diagrama de casos de Uso

Na figura 3.2 está representado o Diagrama de casos de uso da aplicação EventTrackingApp. Neste são representadas todas as acções que um utilizador pode realizar na mesma. Os principais casos de uso desta aplicação são:

- Fazer o Registo, através da opção *Register*;

- Fazer o login, através da opção *login*, podendo também usar o login pelo provedor da *Google*;

- Criar um Evento, usando a opção *Create Event*;

- Usar a sala de conversa, usando a opção *Chat*;

- Consultar o mapa e as localizações dos utilizadores no mapa, opção *Map*;

- Adicionar imagens ao Mapa, usando opção *Add Image*;

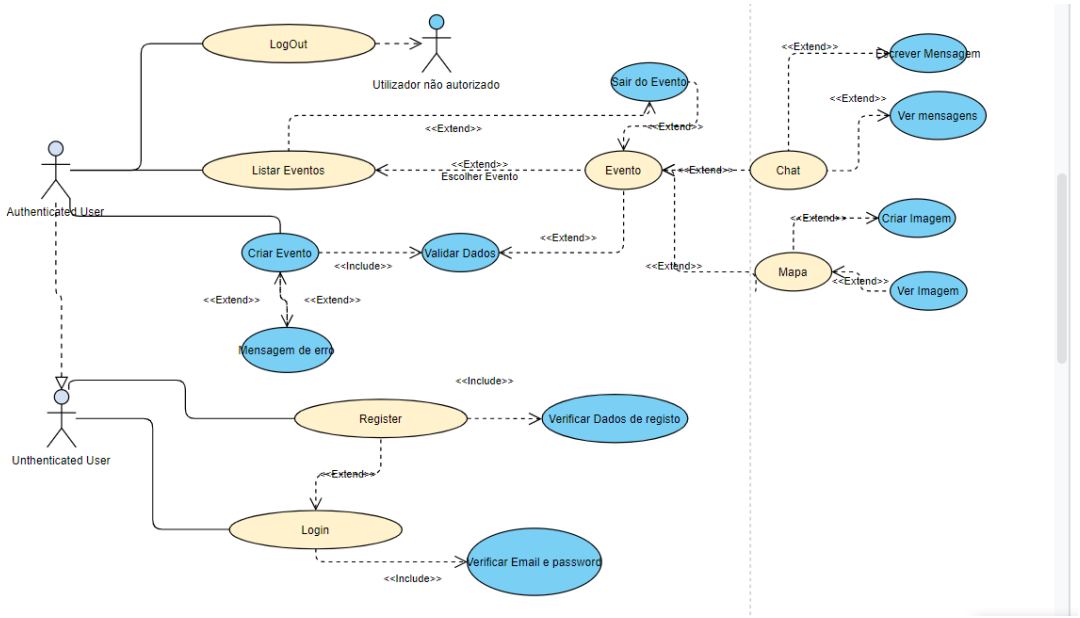

Figura 3.2: Diagrama de casos de uso da aplicação

3.6 Diagrama de Classes

Na figura 3.3 está representado o Diagrama de Classes da aplicação *EventTrackingApp*. O Diagrama de classes mostra as relações entre classes, os seus atributos e operações.

A Classe Utilizador Autenticado contém os seguintes atributos,*Username*, *password* onde é guardado o nome do utilizador e a password respetivamente, *Email*, *user_Id* que é um identificador único do Utilizador e por fim a *Image_url* que serve para guardar o Uniform Resource Locator (URL) da imagem do mesmo, esta imagem é guardada no *fireBase Storage*. Esta Classe tem bastantes operações como *criar Evento()* esta operação cria um evento ficando este utilizador como autor do Evento podendo este evento ser acedido por qualquer utilizador autenticado na aplicação. Operação *Eliminar Evento()* elimina o objeto da classe *Evento*, ao eliminar um Evento são eliminados os objetos que têm uma relação de dependência com esta Classe, como os objetos da classe *Position*, *Image* e *Message*. A operação *Sair do Evento()* permite ao utilizador deixar um evento, não sendo a sua posição mais visível para os outros participantes do Evento. A operação *LogOut()* permite ao utilizador deixar de estar autenticado na aplicação. A operação *Adicionar Imagem()* permite a um utilizador adicionar uma imagem a um Evento e *Eliminar Imagem()* respectivamente eliminar uma Imagem. A operação *Fechar Evento()* permite fechar um evento.

A Classe Evento serve para guardar a informação de um Evento, contendo os seguintes atributos, *Event Name*, *Date* data em que ocorre o Evento, *City* cidade onde ocorre o evento, *Street* rua do evento, *EventId*, *owner* identifica o utilizador que criou o Evento, *Description* breve descrição do Evento, *isClosed*.

Classe *Images*, esta classe representa uma imagem guardada por um utilizador autenticado em um determinado Evento e por isso esta classe tem os seguintes atributos *EventID* que guarda o ID do evento e *user_id* atributo que indica o id do user que criou a imagem, como também *GeoPoint* (latitude e longitude),*imageName*, *imageUrl*, *Description*, *imageUrl* caminho para a imagem guardada no *Firebase Storage* e *timestamp*.

A classe *Position* tem dois atributos, *GeoPoint* e *time*. Esta só existe se existir um utilizador autenticado, podendo um utilizador estar relacionado com vários ojetos da classe *Position* pois estas posições são guardadas para que depois possa ser desenhado o seu percurso em determinado Evento.

A classe *Message* tem os seguintes atributos, *EventId*, *messageBody*, *sender* e *time*. Esta classe só existe se existir um Evento, podendo um Evento ter vários objetos do tipo *Message*.

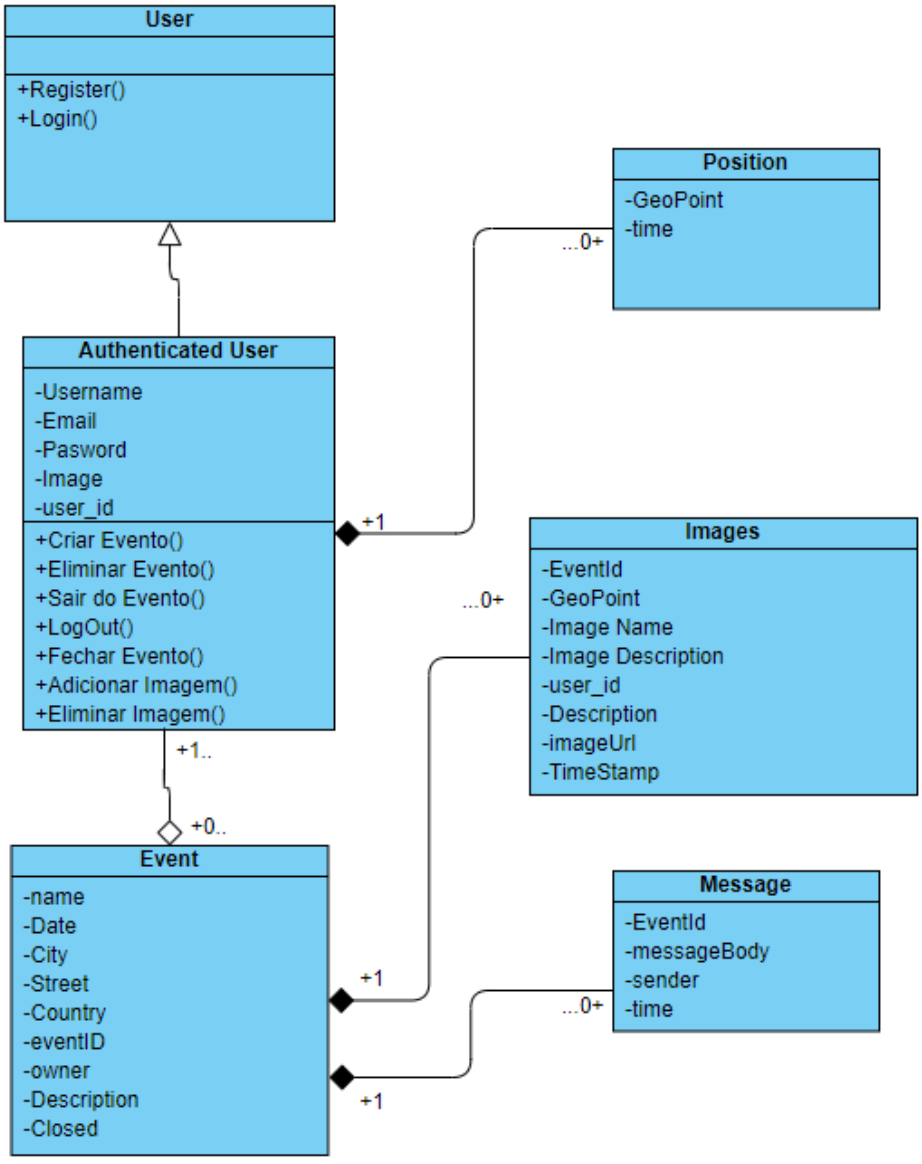

Figura 3.3: Diagrama de Classes

3.7 Diagrama de Atividades

Nesta secção são apresentados os diagramas de atividade das principais funcionalidades da aplicação.

Na figura 3.4 é apresentado o diagrama da atividade Registo.

Na figura 3.5 é apresentado o diagrama da atividade Login.

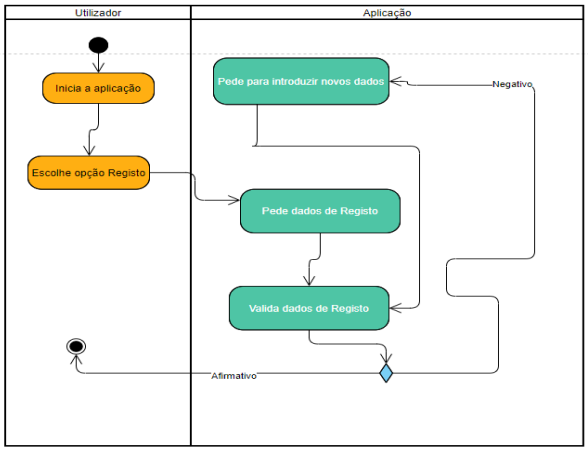

Figura 3.4: Diagrama da atividade Registo

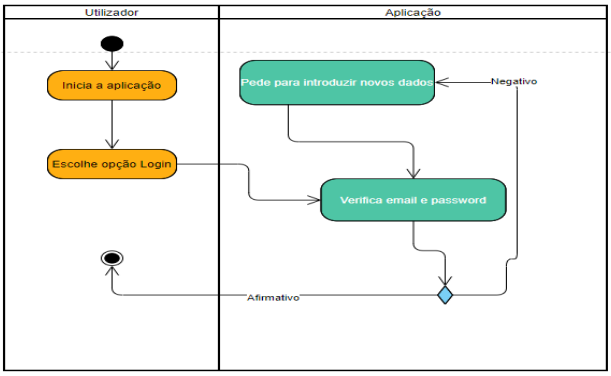

Figura 3.5: Diagrama da atividade Login

Na figura 3.6 é apresentado o diagrama da atividade Criar Evento.

Na figura 3.7 é apresentado o diagrama da atividade Adicionar Imagem.

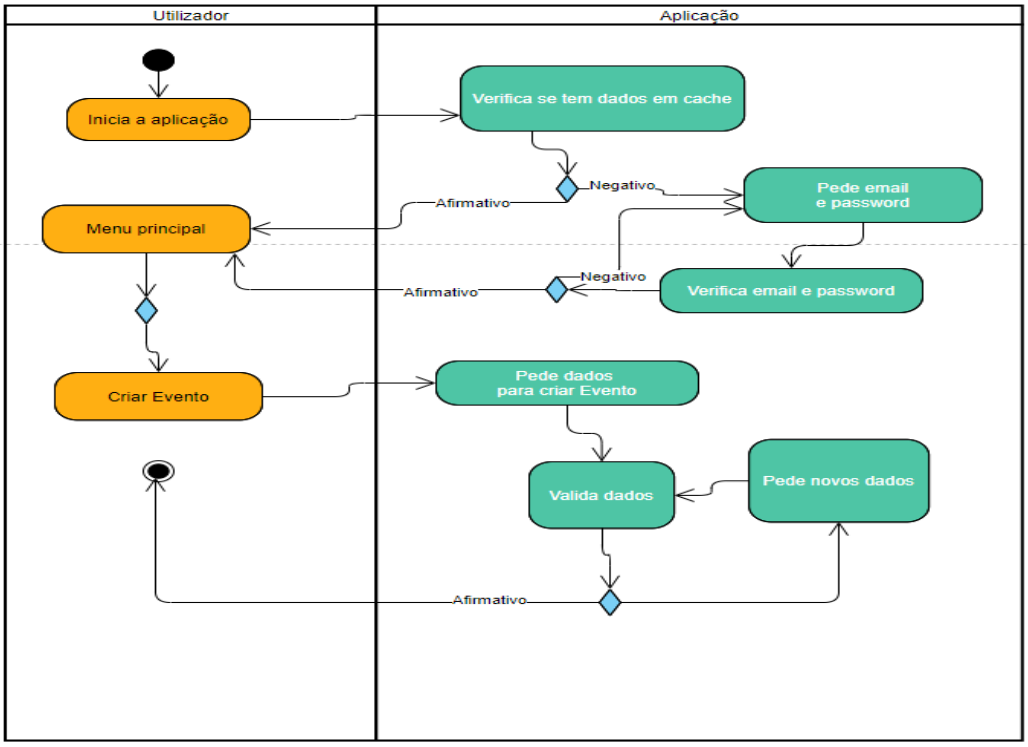

Figura 3.6: Diagrama da atividade Criar Evento

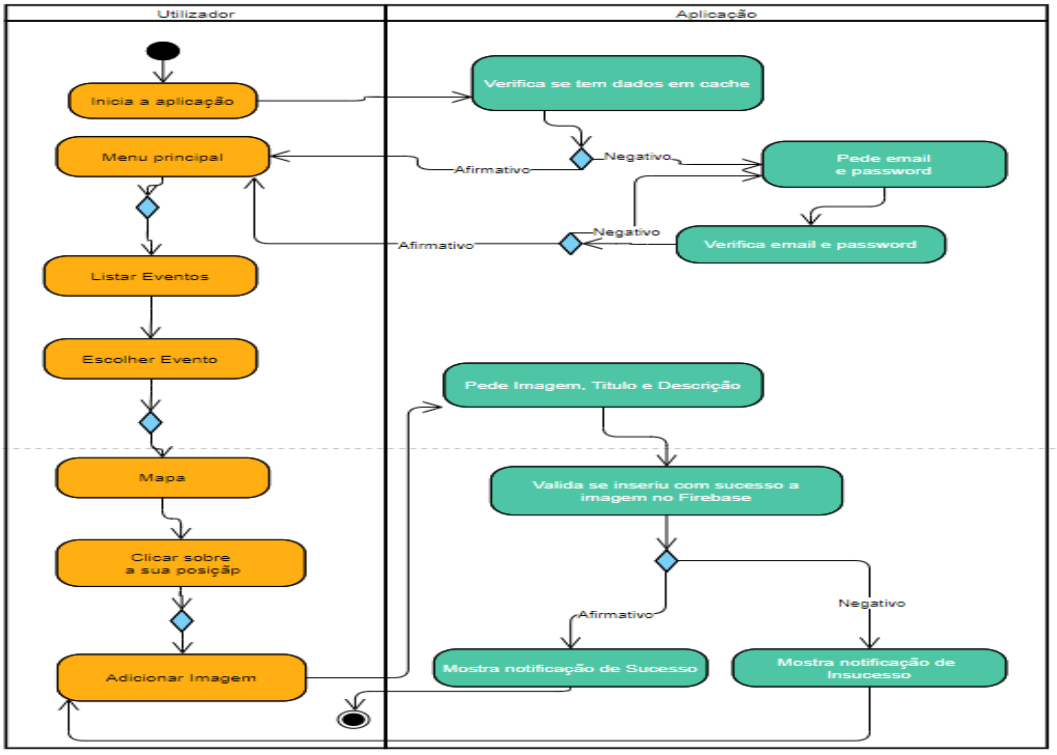

Figura 3.7: Diagrama da atividade Adicionar Imagem

Na figura 3.8 é apresentado o diagrama da atividade Enviar Mensagem.

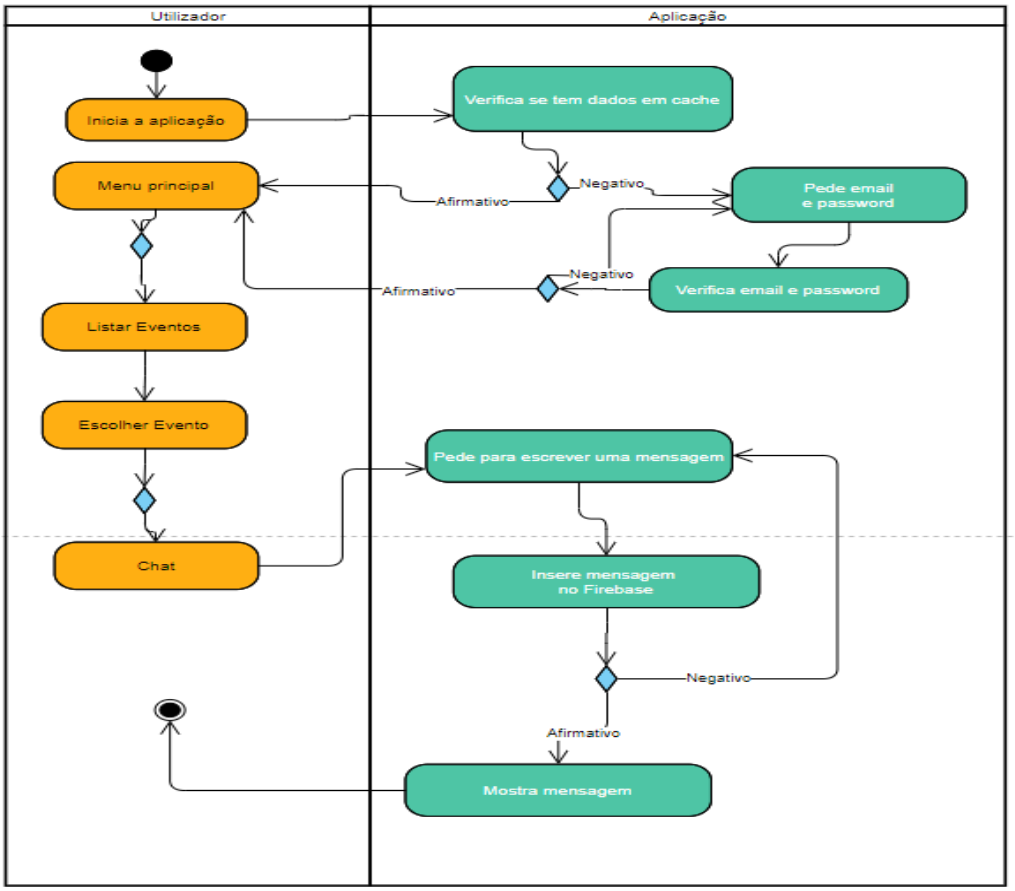

Figura 3.8: Diagrama da atividade Enviar Mensagem

Capítulo 4

Tecnologias e Ferramentas

4.1 Introdução

Na presente secção, serão discutidas as diversas ferramentas e tecnologias usadas para o desenvolvimento da aplicação *EventTrackingApp*.

4.2 Tecnologias utilizadas

4.2.1 Java

Java é uma linguagem de programação desenvolvida na década de 90 por uma equipa de programadores chefiada por *James Gosling*, na empresa *Sun Microsystems*. Tendo depois sido adquirida pela *Oracle Corporation* em 2008.

Sendo esta uma linguagem interpretada, ou seja, o código fonte é executado por um computador (interpretador) e só depois é executado pelo sistema operativo.

A linguagem Java é compilada para bytecode e depois interpretada por uma máquina virtual, sendo esta ainda uma das linguagens mais importantes para o desenvolvimento para a plataforma *Android*.

4.2.2 XML

Extensible Markup Language (XML) é uma linguagem de marcação recomendada pela World Wide Web Consortium (W3C)[9] para a criação de documentos com dados

organizados hierarquicamente, tais como textos, base de dados ou desenhos vetoriais. Sendo esta uma linguagem extensível porque permite gerar novos elementos de marcação. XML é uma derivação do SGML*[10], sendo este um padrão internacional para definir uma linguagem de marcação.

XML foi usado para construir a parte da Interface do utilizador (UI) da aplicação *EventTrackingApp*, tendo sido utilizado para projetar os *layouts* de todos os ecrãs desta aplicação.

4.2.3 GPS

O GPS começou a ser desenvolvido em 1973, ficando operacional em 1974. O sistema foi desenvolvido pelo departamento de defesa dos Estados Unidos[11]. Tendo sido desenvolvido com fins militares, sendo depois disponibilizada para uso público, apesar de ter algumas limitações de precisão.

O GPS é constituído por três partes[12]:

- Satélites - Os satélites circulam em torno da Terra, demorando 12 horas a completar uma volta. Existem 24 satélites dos quais 3 são suplentes.

- Controlo de solo - São estações de monitorização presentes na Terra. As actividades de controlo incluem rastrear e operar os satélites no espaço e monitorizar as suas transmissões.

- Receptores - Os receptores podem ser smartphones, relógios, computadores, etc.

Para receber a localização no espaço bidimensional é necessário apenas 3 satélites, no caso da posição em três dimensões, latitude, longitude e elevação, seria necessário 4 satélites[13]. O quarto satélite não serve só para ajudar a calcular a distância como também ajuda a ajustar o tempo entre os satélites e o *smartPhone*, visto que os *smartPhones* usam relógios de *Quartzo* e não *Atómicos* como os satélites[14].

O GPS foi utilizado na aplicação para receber a localização do utilizador, tendo sido este fundamental para que fosse possível fazer o seguimento de todos os utilizadores de um Evento.

4.3 Ferramentas

4.3.1 Android Studio

Android studio [15] é um IDE oficial para o desenvolvimento de aplicações Android, sendo baseado no *IntelliJ IDEA*. Oferecendo um editor de código e um conjunto de ferramentas da *IntelliJ*.

Fornece também um compilador e um emulador, facilitando os testes sobre a aplicação, como também fazer debug da mesma.

4.3.2 Firebase

Firebase é uma plataforma de desenvolvimento de aplicações do tipo Backeend-as-a-Service (BaaS) que disponibiliza vários serviços tendo como principais, os seguintes:

- realtime database - base de dados não relacional em tempo real;
- cloud Storage - armazenamento em nuvem;
- Firebase Authentication - sistema de autenticação;
- Cloud Firestore - base de dados não relacional baseada em colecções e documentos.

4.3.2.1 Cloud Firestore

Cloud Firestore é uma base de dados não relacional que fornece actualizações em tempo real, ao acontecer uma mudança de estado. Sendo possível criar "listeners" que ficam à escuta de alterações no estado dos dados e emitem um evento para a aplicação informando que os dados foram modificados, facilitando o processo de manter os dados que mostramos e usamos na nossa aplicação atualizados.

Apesar do Cloud Firestore ser *NOSQL*, este suporta transacções *ACID* através da utilização de transacções em lote[16]. Com esta abordagem podemos fazer um série de transacções e se uma destas transacções falhar, é feito um *rollback* fazendo com que o estado antigo da base de dados seja restaurado.

4.3.2.2 Cloud Storage

Cloud Storage permite armazenar ficheiros, esta base de dados permite praticamente alocar a quantidade que queremos sendo massivamente escalável, conseguindo armazenar toda a informação mesmo no caso da aplicação ser bastante grande, por exemplo o *spotify* utiliza a cloud da google para guardar os seus dados na mesma infraestrutura que o cloud storage[17]. No meu caso o *Cloud Storage* foi usado para guardar as imagens criadas pelo utilizador para ver no mapa, como também as imagens que representam os ícones dos utilizadores.

4.3.2.3 Firebase Authentication

O Firebase Authentication é encarregado de restringir o acesso aos dados, tornando obrigatório a autenticação para um utilizador poder aceder aos mesmo. O Firebase Authentication fornece autenticação usando email/senha ou com provedores de identidade federados como Google, Twitter, Facebook e outros.

Na aplicação *EventTrackingApp* é possível fazer login com o provedor da Google[18].

Capítulo 5

Desenvolvimento

5.1 Introdução

Este capítulo aborda a implementação da aplicação, explicando a utilização da base de dados escolhida, como foi implementada a API da *FireBase* sobre *Android*, usando vários excertos de código e explicando os devidos excertos de código.

5.2 Implementação

5.2.1 Primeira solução de Base Dados e servidor

A primeira solução escolhida foi usar o servidor em apache disponibilizado pela Universidade da Beira Interior usando para isto Hypertext Preprocessor (PHP) e para base de dados usar o *MYSQL*, depois de ter feito algum código e alguns testes, conclui que para o meu projeto não necessitava ter o meu próprio servidor. Tendo realizado uma análise sobre os diversos servidores na nuvem acabando por decidir usar o *FireBase*.

Escolhi o *FireBase* pois este fornece todos os serviços que necessitaria para a minha aplicação, como Autenticação, serviço de armazenamento de ficheiros e um serviço de base de dados na nuvem.

O *FireBase* disponibiliza como base de dados duas opções *realtime database* e *Cloud Firestore*. Para esta aplicação utilizei o *Cloud Firestore*, este permite uma maior organização dos dados, sendo estes guardados numa estrutura constituída por coleções de documentos. E principalmente na forma como se hierarquia os dados, sendo

esta hierarquia bastante intuitiva e fácil de escalar. Por exemplo na nossa aplicação, temos uma coleção chamada Eventos, em que cada coleção guarda a informação de um Evento e dentro desta coleção temos inserida uma coleção *Users* que contém todos os utilizadores pertencentes a esse Evento, sendo cada *User* um documento. O que torna fácil consultar todos os utilizadores desse Evento pois bastaria fazer uma consulta e ir buscar todos os Utilizadores (documentos) inseridos em determinado Evento.

O *realtime Database* armazena os seus dados como uma grande árvore, sendo os dados, objetos JavaScript Object Notation (JSON), isto faz com que gerir e manter esta base de dados se torne numa tarefa com uma maior complexidade sendo preciso manter a base de dados escalável. Se a árvore não estiver bem ramificada pode ser preciso percorrer quase a árvore toda só para consultar um dado. Este problema não acontece na *Cloud Firestore* pois os dados estão divididos em coleções, o que torna a sua organização mais intuitiva e simples. Exemplo da estrutura usada no Firebase na figura 5.1;

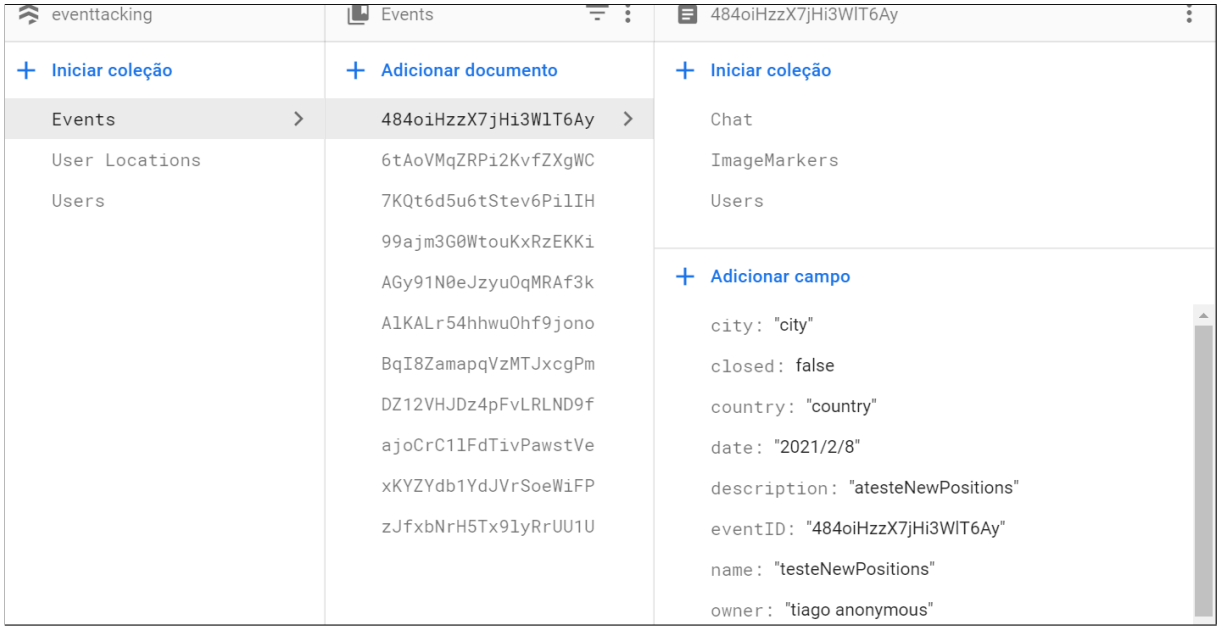

Figura 5.1: Estrutura da aplicação no *Firebase*

5.2.1.1 Leitura de Dados a partir da Cloud

Na aplicação *EventTrackingApp* é necessário em algumas situações estar sempre a verificar se existe alguma alteração nos dados, estado da base de dados, como por exemplo se um utilizador estiver no ecrã do Chat é importante que a última mensagem mostrada seja a última que existe na base de dados. Para isto podemos usar um listener, foi usada uma função disponibilizada pelo *Firebase* chamada addSnapShotListener esta função cria um listener, este listener pode estar ligado a uma coleção ou a um conjunto de documentos.

Quando existe uma alteração no estado dos dados, o listener despoleta um evento, utilizando depois o evento para atualizar os dados na aplicação, mantendo assim a informação presente na aplicação sempre atualizada.

É importante não esquecer de terminar estes Listeners, de forma a limpar os recursos utilizados pelos mesmos e mais importante estes podem causar erros na aplicação, tentando atualizar informação na aplicação podendo esta informação já nem existir na mesma. Isto pode ser feito de duas formas:

- Guardando uma instância do listener e quando o quisermos terminar apenas temos de chamar esta instância antes guardada, fazendo instância.remove() (ver figura 5.2);

```
protected void onDestroy() {
    super.onDestroy();
    listenerImages.remove();
    listenUserPositions.remove();
    listenUsers.remove();

}
```

Figura 5.2: Remover instâncias

- Outra forma de terminar um listener, também disponibilizada pelo Firebase, chamando addSnapShotListener esta função aceita como parâmetro uma variável que pode ser usada para controlar o listener, por exemplo na nossa aplicação

este método foi usado para parar de escutar novas mensagens ao sair do ecrã do chat, enviando como parâmetro a actividade Chat, isto significa que ao terminar a actividade do chat, é terminado por consequente o listener, como ilustrado na figura 5.3.

```
protected void getMessagesFromServer() {
    mDb.collection( collectionPath: "Chat")
            .whereEqualTo( field: "eventId", eventID).orderBy("time")
            .addSnapshotListener( activity: ChatActivity.this, (value, e) → {
                    if (e != null) {
                        Log.w(TAG,  msg: "Listen failed.", e);
                        return;
                    }
                    ListView myListView = (ListView) findViewById(R.id.messages_view);
                    ArrayList<Message> messageList = new ArrayList<>();
                    for (QueryDocumentSnapshot doc : value) {

                        if (doc.get("messageBody") != null && doc.get("sender") != null  && doc.get("time") != null) {
                            boolean sendByUs = doc.get("sender").equals(currentUser);
                            boolean isAdmin = doc.get("sender").equals(session.getEvent().getOwner());
                            Message message = new Message(doc.get("sender").toString(), doc.get("messageBody").toString(),
                                    sendByUs, doc.get("eventId").toString(), doc.get("time").toString(), isAdmin);
                            messageList.add(message);
                        }
                    }
                    MessageAdapter adapter = new MessageAdapter( context: ChatActivity.this, resource: 0, messageList);
                    myListView.setAdapter(adapter);
            });
```

Figura 5.3: Remover instância enviando a actividade

Na figura 5.4 é usado o onCompleteListener uma vez que não é preciso ficar a escuta de alterações no evento, em caso de sucesso retorna os dados do evento, em caso de insucesso faço um log a informar que ocorreu um erro a ler o documento, este erro não é visível para o utilizador.

```
private void getEventInfo() {
    DocumentReference docRef = mDb.collection( collectionPath: "Events").document(eventID);
    docRef.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
        @Override
        public void onComplete(@NonNull Task<DocumentSnapshot> task) {
            if (task.isSuccessful()) {
                DocumentSnapshot document = task.getResult();
                if (document.exists()) {
                    Log.d(TAG,  msg: "DocumentSnapshot data: " + document.getData());

                    Event event = new Event(document.get("owner").toString().trim(),
                            document.get("eventName").toString().trim(),
                            document.get("description").toString().trim(),
                            document.get("street").toString().trim(), document.get("city").toString().trim(),
                            document.get("country").toString().trim(),document.get("eventChoosenDate").toString().trim(),
                             isClosed: document.get("isClosed") != null && (boolean)document.get("isClosed") );
                    setInfo(event);
                } else {
                    Log.d(TAG,  msg: "No such document");
                }
            } else {
                Log.d(TAG,  msg: "get failed with ", task.getException());
            }
        }
    });
```

Figura 5.4: Buscar informação de um Evento

5.2.1.2 Escrita de Dados na Cloud Firestore

Para escrever dados na *Cloud FireStore* o *Firebase* disponibiliza a função update que como parâmetro podemos enviar um objeto ou também por exemplo HashMap. O HashMap no meu caso foi usado para modificar um documento e não substituir o documento ou todos os dados nele, assim apenas vou alterar, adicionar os campos novos que enviar no HashMap mantendo os campos antes inseridos nesse documento.

Para saber quando os dados já foram inseridos é chamado o addOnCompleteListener. Na figura 5.5 é feito uma escrita para a *Cloud FireStore*, neste caso como só queremos modificar um campo na classe Evento, é enviado um HashMap com a chave *isClosed*.

```
private void closeEvent() {
 DocumentReference eventDocument =   mDb.collection( collectionPath: "Events").document(eventID);
    HashMap<String,Object> closeEventHashMap = new HashMap();
    closeEventHashMap.put("isClosed", true);
    eventDocument.update(closeEventHashMap).addOnCompleteListener((task) → {
            if (task.isSuccessful()) {
                Log.d(TAG,  msg: "Update Successful");
                Toast.makeText( context: EventMainCopy.this,  text: "Event Closed with success",
                        Toast.LENGTH_SHORT).show();
                btn_closeEvent.setVisibility(View.INVISIBLE);
                session.getEvent().setClosed(true);
            } else {
                Log.w(TAG,  msg: "Error updating document", task.getException());
            }
    });
```

Figura 5.5: Fechar um evento

5.2.1.3 Atualizar posições dos participantes de um Evento

Para que o utilizador possa visualizar as posições de todos os participantes de um evento no mapa e para que estas posições sejam as posição actuais de cada participante, é preciso verificar as posições de cada participante. Para isto decidi usar um *handler* que a cada 3s verifica as posições de cada participante do Evento e actualiza a sua posição no mapa, no caso de a nova posição ser diferente da posição actual.

Eu optei por esta abordagem porque por exemplo se optasse por usar um listener às posições dos participantes de um evento, isto poderia ter muitas alterações em um pouco espaço de tempo fazendo com que a aplicação fica-se lenta ou fazendo mesmo

com que ela crasha-se. Com esta abordagem posso controlar o tempo que queremos para que se actualize as posições, uma desvantagem desta abordagem é que podemos fazer leituras desnecessárias à base de dados. Isto é demonstrado nos dois seguintes excertos de código.

Na função *retrieveUserLocations* a flag *hasChanges* valida a posição nova do utilizador que recebi do *Firebase* com a posição antiga que está no marcador do utilizador isto para validar se as duas posições forem iguais não precisar de atualizar o marcador e posso seguir para o próximo utilizador.

```
  private static final int LOCATION_UPDATE_INTERVAL = 3000;
    private void startUserLocationsRunnable(){
     isUserLocationRunning = true;
     mHandler.postDelayed(mRunnable = new Runnable() {
         @Override
         public void run() {
             retrieveUserLocations();
             mHandler.postDelayed(mRunnable, LOCATION_UPDATE_INTERVAL);
         }
    }, LOCATION_UPDATE_INTERVAL);
 }
```

```
private void retrieveUserLocations(){
    try{
    for(final MyClusterItem clusterItem: mClusterItems){
        DocumentReference userLocationRef = FirebaseFirestore
        .getInstance()
                .collection("User Locations")
                .document(clusterItem.getUser().getUser_id());

        userLocationRef.get().addOnCompleteListener
        (new OnCompleteListener<DocumentSnapshot>() {
            @Override
            public void onComplete(@NonNull Task<DocumentSnapshot> task)
            {
                if(task.isSuccessful()){
                    final UserLocation updatedUserLocation =
                            task.getResult().toObject(
```

```
                        UserLocation.class);
                // update the location
                for (int i = 0; i < mClusterItems.size(); i++) {
                    try {
                        if (mClusterItems.get(i).getUser().
                                getUser_id().equals(
                                updatedUserLocation.getUser()
                                .getUser_id())) {

                            LatLng updatedLatLng = new LatLng(
                                    updatedUserLocation.getGeoPoint()
                                    .getLatitude(),
                                    updatedUserLocation.getGeoPoint()
                                    .getLongitude()
                            );
                            boolean hasChanges =
                            !mClusterItems.get(i).getPosition()
                                    .equals(updatedLatLng);
                            if((drawFirstTime || hasChanges) &&
                            mClusterItems.get(i).getUser().
                                    getUser_id().equals(session.
                                    getUser().getUser_id())) {
                                updateMyCurrentPosition();
                                drawFirstTime = false;
                            }
                            if(!hasChanges) continue;
                            mClusterItems.get(i).setPosition(
                            updatedLatLng);
                            clusterManagerRenderer.setUpdateMarker(
                            mClusterItems.get(i));

                        }
                    }
                }
            }
        }
    });
  }
  ...
}
```

5.2.1.4 Serviço para atualizar posição do utilizador

Para que a posição do utilizador esteja sempre actualizada na base de dados, foi criado um serviço *LocationTrackingService* (excerto de código 5.1) neste serviço uso a classe *FusedLocationProviderClient* esta classe extende uma API da Google que permite obter a posição do utilizador, utilizando também a função requestLocationUpdate que recebe como parâmetros um objecto da classe LocationRequest onde posso definir várias opções como o intervalo com que recebo a minha localização ou definir uma distância mínima percorrida para receber outra localização. RequestLocationUpdate chama recursivamente a função onLocationResult.

O LocationRequest recebe vários parâmetros:

- *setInterval* este é o intervalo com que o serviço pede localizações neste caso é de 5 em 5 segundos;
- *setFastestInterval* este é o intervalo mínimo com que o serviço recebe actualizações de localização, este tempo é usado quando outra aplicação está a usar o serviço de localização e a nossa aplicação pode também utilizar esse valor recebido, neste caso o valor é de 2 segundos;
- *setSmallestDisplacement* esta é a diferença de distância mínima que é preciso para haver outra atualização de localização, neste caso este valor é de 5 metros;

Informação baseada na documentação da *Google Android*[19].

```
public class LocationTrackingService extends Service {
...
    private FusedLocationProviderClient mFusedLocationClient;
    private final static long UPDATE_INTERVAL = 5 * 1000;  /* 5 secs */
    private final static long FASTEST_INTERVAL = 2 * 1000; /* 2 sec */
    private final static int MIN_DISTANCE = 5; /* 5 meters */
    private void getLocation() {

        // ---------- LocationRequest ----
        // Create the location request to start receiving updates
        LocationRequest mLocationRequestHighAccuracy =
        new LocationRequest();
        mLocationRequestHighAccuracy.setPriority(
        LocationRequest.PRIORITY_HIGH_ACCURACY);
        mLocationRequestHighAccuracy.setInterval(UPDATE_INTERVAL);
```

```
        mLocationRequestHighAccuracy.setFastestInterval(
        FASTEST_INTERVAL);
        mLocationRequestHighAccuracy.setSmallestDisplacement(
        MIN_DISTANCE);

            .
            .
            .

        mFusedLocationClient.requestLocationUpdates(
        mLocationRequestHighAccuracy, new LocationCallback() {
                    @Override
                    public void onLocationResult(
                    LocationResult locationResult) {

                        Location location = locationResult.
                        getLastLocation();

                        if (location != null && !session.getEvent().
                        isClosed()) {
                            User user = session.getUser();
                            session.setCurrentLocation(location);

                            CustomGeoPoint geoPoint =
                            new CustomGeoPoint(location.getLatitude(),
                            location.getLongitude());
                            UserLocation userLocation =
                            new UserLocation(geoPoint, null, user);
                            saveUserLocation(userLocation);
                            addUserPosition(user, session.getEvent().
                            getEventID(),
                            location, geoPoint);
                        }
                    }
                },
                Looper.myLooper());
    }
}
```

Listing 5.1: Excerto de código: Serviço de Tracking

Dentro da função *onLocationResult* guardo a nova posição do utilizador, como

também as posições do utilizador em relação ao evento, este último ponto é para desenhar no mapa o trajeto percorrido pelo utilizador no Evento.

Na função *saveUserLocation* (excerto de código 5.2) chama *addOnCompleteListener*, da API da FireBase, para adicionar a nova posição do utilizador à coleção *User Locations*.

Modificando o documento antigo que contém a penúltima posição do utilizador pela nova posição do utilizador.

```
private void saveUserLocation(final UserLocation userLocation) {

    try{
        DocumentReference locationRef = FirebaseFirestore.getInstance()
                .collection(
                getString(R.string.fire_store_users_locations))
                .document(FirebaseAuth.getInstance().getUid());

        locationRef.set(userLocation).addOnCompleteListener(
        new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if(task.isSuccessful()){
                    Log.d(TAG, "onComplete:
                    LocationInserted");
                }
            }
        });
    }catch ...

}
```

Listing 5.2: Excerto de código:Guardar nova posição

Para mostrar o trajeto percorrido por um utilizador é necessário guardar todas as posições desse utilizador, para isso criei a função *addUserPosition* (excerto de código 5.3) que também é chamada no *LocationTrackingService*, esta função chama o onCOmpleteListener do *Firebase* para adicionar a nova posição à coleção *UserPositionsInEvent* que contém todas as posições daquele utilizador em determinado Evento, é também guardado o tempo atual em milissegundos para que depois o trajeto possa recolher as posições por ordem cronológica.

Experimentei guardar em segundos mas isto fez com que algumas posições não ficassem bem ordenadas pois o utilizador poderia estar a andar rápido por exemplo se estivesse de carro ou bicicleta.

```
private void addUserPosition(final User user, final String EventID,
final Location location, final CustomGeoPoint geoPoint) {
    final String key =  user.getUser_id() + '_' + EventID;
     FirebaseFirestore.getInstance().collection(
     USERPOSITIONSINEVENT).whereEqualTo(USERPOSITIONKEY, key).
     get().addOnCompleteListener(
     new OnCompleteListener<QuerySnapshot>() {
             @Override
                public void onComplete(@NonNull Task<QuerySnapshot>
                task) {
                    if(task.getResult().getDocuments().size()
                    != 0 ) {
                        DocumentReference documentReference =
                        task.getResult().getDocuments().
                        get(0).getReference();
                        DocumentSnapshot document = task.
                        getResult().getDocuments().get(0);
                        if(checkUserMoved(document, location)) {
                            Long tsLong = System.currentTimeMillis();
                            UserPosition customGeoPoint  =
                            new UserPosition(location.getLatitude(),
                            location.getLongitude(), tsLong.toString());
                            documentReference.collection(
                            USERPOSITION).add(customGeoPoint);
                            documentReference.update("lastPosition",
                            convertLocationToCustomGeoPoint(location));
                            updateDistanceTraveled(
                            documentReference, document, location);
                        }
                    }else {
                        UserLocationPositionsInEvent docInfo =
                        new UserLocationPositionsInEvent(
                        key, 0, geoPoint);
                        FirebaseFirestore.getInstance().collection(
                        USERPOSITIONSINEVENT).add(docInfo).
                        addOnCompleteListener(
                        new OnCompleteListener<DocumentReference>() {
                            @Override
```

```
                        public void onComplete(
                        @NonNull Task<DocumentReference> task) {
                            Long tsLong = System.currentTimeMillis();
                            UserPosition customGeoPoint =
                            new UserPosition(location.getLatitude(),
                            location.getLongitude(),
                            tsLong.toString());
                            task.getResult().collection(
                            USERPOSITION).add(customGeoPoint);
                        }
                    });
                }
            }
        });

}
```

Listing 5.3: Excerto de código:Guardar posição na coleção UserPositionsInEvent

5.2.2 Google Maps API

Como é necessário mostrar posições de utilizadores e de imagens no mapa, foi utilizado o *Google Maps* API para mostrar o mapa tal como as posições dos utilizadores e as imagens.

O primeiro ponto é necessário pedir permissão para o *Google* ter acesso a localização do *smartPhone* (excertos de código 5.4 e 5.5).

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

Listing 5.4: Excerto de código: Permissão aceder localização

```
private void getLocationPermission() {

if (ContextCompat.checkSelfPermission(this.getApplicationContext(),
        android.Manifest.permission.ACCESS_FINE_LOCATION)
        == PackageManager.PERMISSION_GRANTED) {
    mLocationPermissionGranted = true;
    isEverythingOK = true;
     getUserDetails();
} else {
```

```
    ActivityCompat.requestPermissions(this,
            new String[]
            {android.Manifest.permission.ACCESS_FINE_LOCATION},
            PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
}
}
```

Listing 5.5: Excerto de código: Função pedir permissão

É necessário também importar uma chave do *Google Maps*, esta chave é utilizada para autenticar os pedidos feitos à API do *Google Maps* (excerto de código 5.6.

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

Listing 5.6: Excerto de código: Chave da API do Google Maps

Para usar o mapa foi criado uma *FragmentActivity* esta é uma extensão da classe *ComponentActivity* mas que suporta Fragmentos, tornando esta classe útil visto que o mapa é um fragmento. Tendo também implementado várias interfaces da google Maps como *OnMapReadyCallback* (ver excerto 5.7) sendo esta utilizada para saber quando o mapa está pronto para ser visualizado e poder fazer operações sobre ele como adicionar as imagens e utilizadores ao mapa.

```
public class MapActivity extends FragmentActivity
implements OnMapReadyCallback, GoogleMap.OnMarkerClickListener...

public void onMapReady(GoogleMap googleMap) {
    mGoogleMap = googleMap;
    googleMap.setOnMapLoadedCallback(new
    GoogleMap.OnMapLoadedCallback() {
        @Override
        public void onMapLoaded() {
            getUsersOfTheEvent();
            getImageMarkers();
            if(isEventClosed) {
                updateMyCurrentPosition();
            }
        }
    });
```

Listing 5.7: Excerto de código: FragmentActivity e interfaces

5.2.3 Carregar e mostrar imagens

Para carregar e mostrar imagens foram usadas duas bibliotecas *Picasso* e *Glide* [20]. Inicialmente comecei apenas por usar o *Picaso*, usando depois o *Glide*[21], este foi usado porque o *Picasso* no ecrã de adicionar uma imagem ao mapa estava a crashar com erro "out of memory", tendo sido isto corrigido com o uso do *Glide*. O Glide apesar de ser uma biblioteca mais pesada previne "out of memory" porque usa menos memoria em cache[22] isto se deve ao facto de o *Glide* carregar as imagens tendo em conta o tamanho da *View*[23] e o *Picasso* carregar a imagem por completo.

Usar *Glide* é bastante simples basta passar o context, url da imagem e a *View* que queremos popular, como ilustra o excerto de código 5.8. O *Thumbnail* significa que a imagem é carregada primeiro com uma resolução menor antes de a imagem ser carregada na totalidade para que se possa mostrar uma *preview* da imagem ao utilizador, neste caso como temos "0.5f", significa que é carregado primeiro a imagem com metade da resolução.

```
Glide.with(getActivity().getApplicationContext())
                        .load(mImageUri)
                        .thumbnail(0.5f)
                        .into(mImageView);
```

Listing 5.8: Excerto de código: Como usar o Glide

Para usar o Picasso primeiro inicializamos o Picasso com algumas definições como o contexto, se queremos ou não usar *memorycache*, neste caso não utilizei por isso passei *Cache.NONE* (mostrado no excerto de código 5.9).

Depois para carregar a imagem basta chamar *Picasso.load*, passando a url da imagem, *resize(80,80)* diz ao Picasso para ajustar a imagem a dimensão enviada por parâmetro neste caso 80 por 80 *pixels*, *centerCrop()* diz ao Picasso para preencher o resto da *View*. Usei um *callBack* para esperar que o Picasso carrega-se a imagem e ao carregar preencher o marker com a imagem carregada.

```
Picasso picasso = new Picasso.Builder(context).executor(
Executors.newSingleThreadExecutor()).memoryCache(Cache.NONE).
indicatorsEnabled(true).build();

picasso.load(item.getIconPicture()).resize(80,80).centerCrop().
into(imageView, new Callback() {
            @Override
```

```
            public void onSuccess() {
                BitmapDrawable drawable = (BitmapDrawable) imageView.
                getDrawable();
                Bitmap bitmap = drawable.getBitmap();
                if(marker != null && marker.getTag() != null) {
                    marker.setIcon(BitmapDescriptorFactory
                    .fromBitmap(bitmap));
                    marker.setVisible(true);
                    extraMarkerInfo.put(marker.getId(), item);
                }
            }

            @Override
            public void onError(Exception e) {
                Log.d("ImageMarkerCluster", e.getMessage());

            }

        });
```

Listing 5.9: Excerto de código: Como usar o Picasso

Capítulo 6

Manual de Utilização e Testes

6.1 Introdução

Neste capítulo será apresentado o manual de utilização da aplicação e os testes efetuados, com a seguinte sequência:

- a secção 6.2 – Manual de Utilização – explica, aos utilizadores da aplicação, o seu funcionamento;

- a secção 6.3 – Testes Efetuados – apresenta os diferentes testes de funcionamento e de performance da aplicação *Android*;

6.2 Manual de Utilização

6.2.1 Login e Registo

A instalação da aplicação num dispositivo Android faz-se de forma semelhante a qualquer outra instalação neste sistema operativo, uma vez que o protótipo foi compilado para um ficheiro `.apk`. Após instalar e iniciar a aplicação é preciso permitir a esta o acesso à localização do dispositivo móvel, como se vê na figura 6.1.

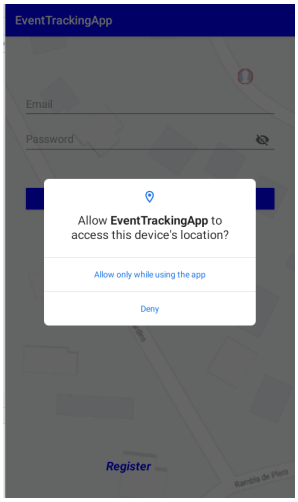

Figura 6.1: Permitir acesso à localização do dispositivo móvel

Depois de permitir o acesso, somos redirecionados para o ecrã de login. Podendo depois efetuar o login na mesma, preenchendo o email e a password com a qual fez o registo (ver figura 6.2). No caso de ainda não ter conta criada, è necessário efectuar assim a criação da própria. Para criar uma nova conta é necessário clicar em Registar.

No ecrã de registar podemos selecionar uma imagem, não sendo obrigatório, no caso de não ser escolhida uma imagem o utilizador fica com uma imagem padrão, escolhendo depois um nome de utilizador, email e password, sendo estes campos obrigatórios (ver figura 6.3).

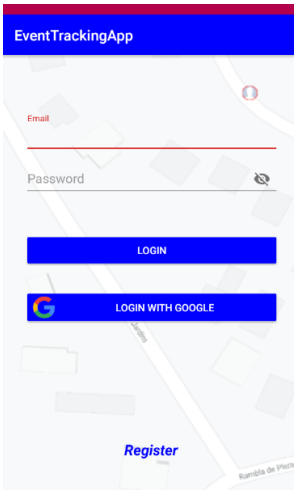

Figura 6.2: Login na aplicação

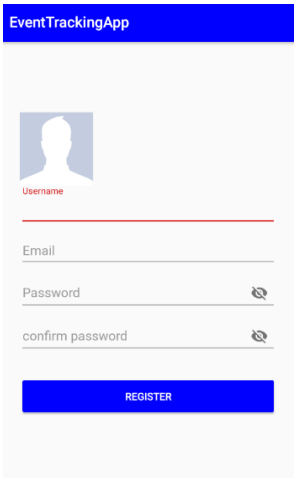

Figura 6.3: Registar na aplicação

6.2.2 Criar Evento

Para criar um Evento é obrigatório preencher o nome, descrição e país, sendo os outros campos facultativos, a data do evento se não for alterada fica com o dia da criação do evento (ver figuras 6.4, 6.5 e 6.6).

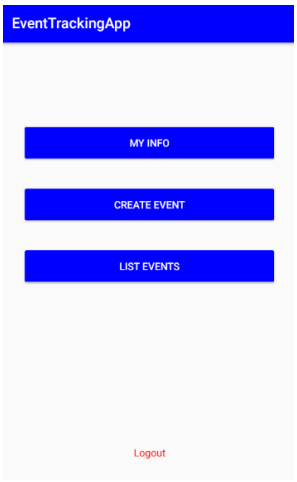

Figura 6.4: Lista opções disponíveis para um utilizador

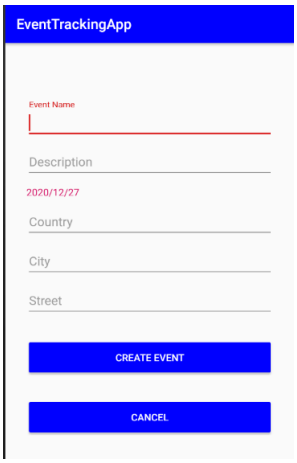

Figura 6.5: Criar Evento

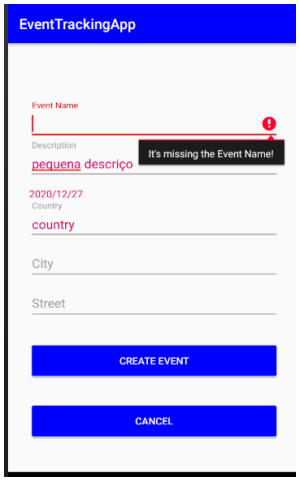

Figura 6.6: Criar Evento com erro

6.2.3 Escolher Evento

No ecrã de listar Eventos é possível filtrar os eventos, por criador do evento ou por nome do evento (figuras 6.7 e 6.8). Para escolher entre criador do evento e nome do evento basta trocar entre as duas opções usando a dropdown.

Tem também a opção de escolher listar apenas os próprios Eventos, para escolher esta opção basta clicar no botão "My Events" (figura 6.9). Carregar no botão "Clear" volta a pesquisar os Eventos sem nenhum filtro.

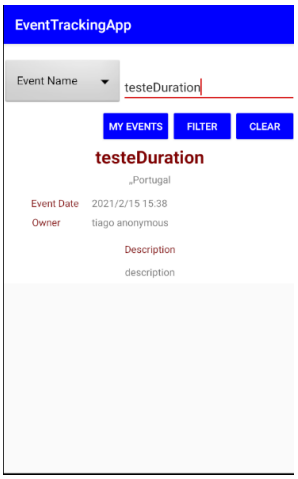

Figura 6.7: Filtrar por nome do Eventos

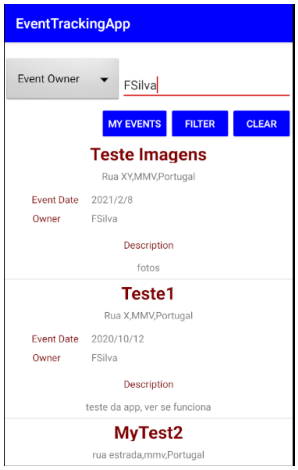

Figura 6.8: Filtrar por proprietário do Evento

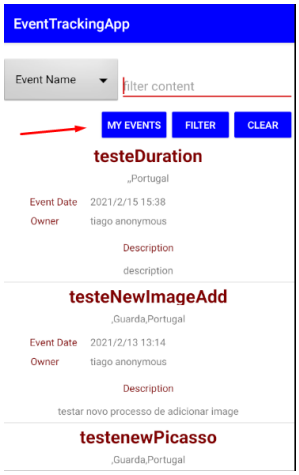

Figura 6.9: Filtrar pelos seus Eventos

6.2.4 Sala de conversa

Para utilizar a sala de conversa de um evento. É preciso selecionar um evento e dentro deste escolher a opção de Chat, sendo depois o utilizador direcionado para a sala de conversa. Nesta o utilizador tem acesso a todas as mensagens partilhadas por todos os utilizadores do Evento, como ilustra a figura 6.11.

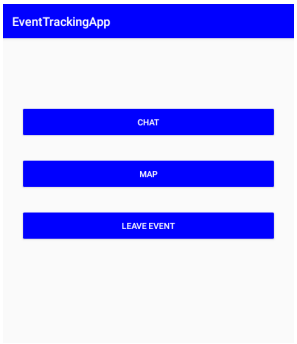

Figura 6.10: Dashboard onde mostra botão para navegar até à sala de conversa

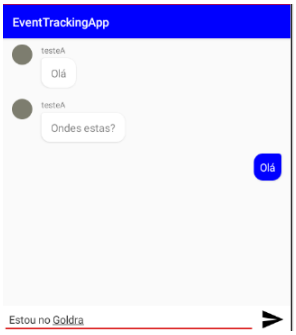

Figura 6.11: Sala de conversa de evento

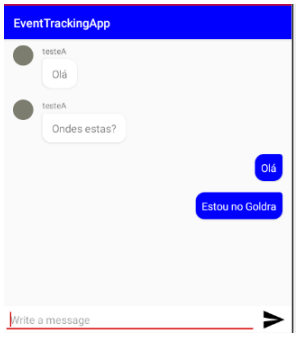

Figura 6.12: Mensagem enviada

6.2.5 Fechar Evento

Fechar um evento faz com que não seja mais possível enviar mensagens nem adicionar imagens, sendo possível visualizar as imagens e mensagens antes postadas (ver figura 6.14). Também não é mais possível ver a posição dos outros participantes do Evento sendo apenas possível ver a própria posição e o trajecto percorrido.

Para fechar um evento é preciso ser o proprietário do evento, podendo ser fechado acedendo ao ecrã principal do evento e depois podemos ver a opção *Close Event* que fecha o evento, como ilustra a figura 6.13.

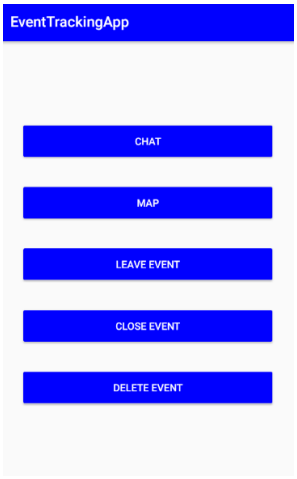

Figura 6.13: Mostra a opção de Fechar Evento (Close Event)

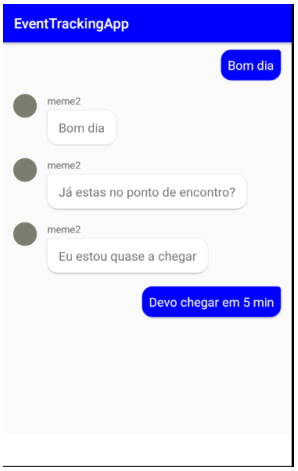

Figura 6.14: Mostra o Chat com a possibilidade de apenas ver as mensagens

6.2.6 Adicionar Imagem

Ao entrar na opção Mapa dentro de um evento, a aplicação foca na posição atual do utilizador. Escolher esta posição mostra várias opções sendo uma delas a opção de adicionar uma imagem. Sendo depois o utilizador direccionado para outro ecrã (adicionar imagem), neste ecrã o utilizador pode escolher uma imagem da sua coleção de imagens das quais armazenadas no seu dispositivo móvel ou na cloud, usando o *Google drive*. Pode escolher também uma descrição para a sua imagem. Esta aplicação apenas permite adicionar uma imagem de cada vez, sendo preciso repetir o processo, para adicionar outra imagem (ver figuras 6.15 a 6.17).

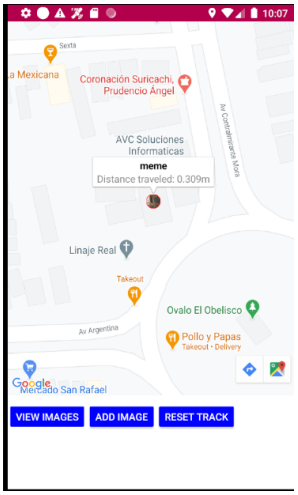

Figura 6.15: Rodapé com a opção de adicionar imagem

Figura 6.16: Ecrã de adicionar imagem

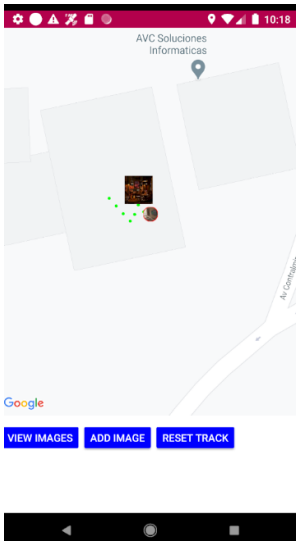

Figura 6.17: Visualizar imagem no mapa

6.2.7 Visualizar várias Imagens e Eliminar uma Imagem

Para visualizar várias imagens basta seguir os mesmo passos para visualizar apenas uma Imagem, escolhendo a opção "View Images" mas depois ao termos várias imagens em uma única posição podemos seleccionar a imagem que queremos visualizar entre todas as imagens presentes nessa posição, escolhendo a aba pretendida na parte superior do *smartPhone* (como podemos ver na figura 6.18). Para eliminar uma imagem basta no ecrã de visualizar imagem seleccionar *"Delete Image"* (ver imagem 6.19).

Na figura 6.20 podemos ver que a segunda imagem foi eliminada e na posição da segunda imagem mostramos a imagem que antes tínhamos na terceira aba.

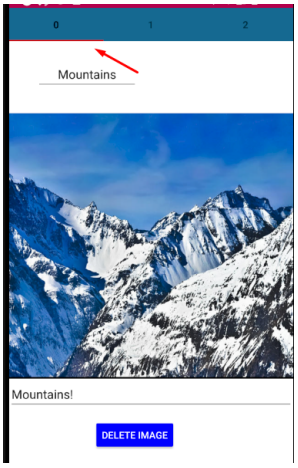

Figura 6.18: Mostra a primeira imagem com a possibilidade de escolher as outras imagens

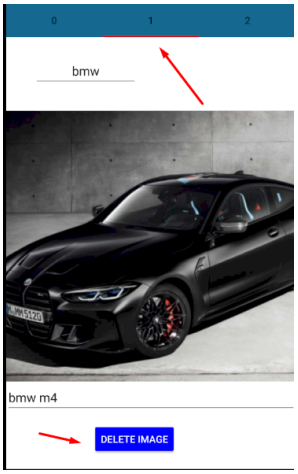

Figura 6.19: Mostra segunda imagem com opção de eliminar imagem

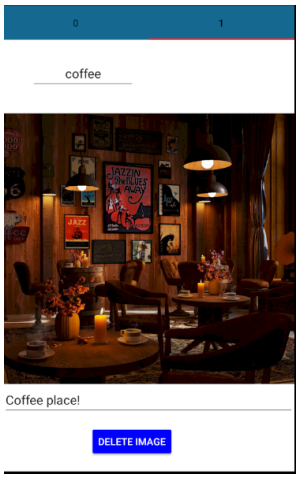

Figura 6.20: Segunda imagem foi substituída pela terceira imagem

6.2.8 Visualizar trajeto e Reiniciar trajeto

Para visualizar o trajeto, simplesmente é preciso escolher o Evento e depois a opção "Map" (figura 6.21), depois a aplicação é direcionada, para o Mapa onde é possível visualizar o trajeto percorrido, como demonstrado na figura 6.22. Para eliminar o trajeto antes percorrido, é necessário fazer clique na marca do utilizador e escolher opção "Reset Track", isto elimina o trajeto e reinicia os dados, como distância, velocidade

e duração (ver figuras 6.23 e 6.24).

A aplicação continua a guardar as posições do utilizador mesmo em segundo plano, utilizando o serviço "TrackingService", como podemos ver na figura 6.25, podendo depois terminar este serviço no caso de querermos parar de atualizar as posições do utilizador (este serviço é terminado no caso de fecharmos por completo a aplicação).

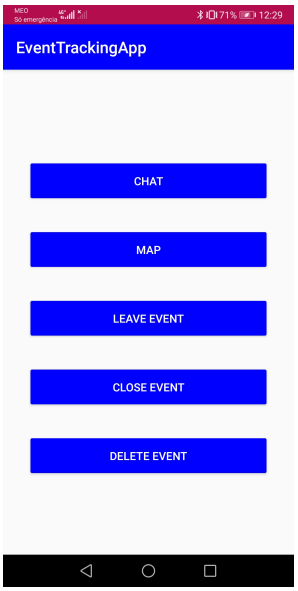

Figura 6.21: Mostra opção "Map"

Figura 6.22: Mostra trajeto depois de escolher ver Mapa

Figura 6.23: Escolher opção "Reset Track" para reiniciar trajeto

Figura 6.24: Percurso depois de reiniciar o trajeto

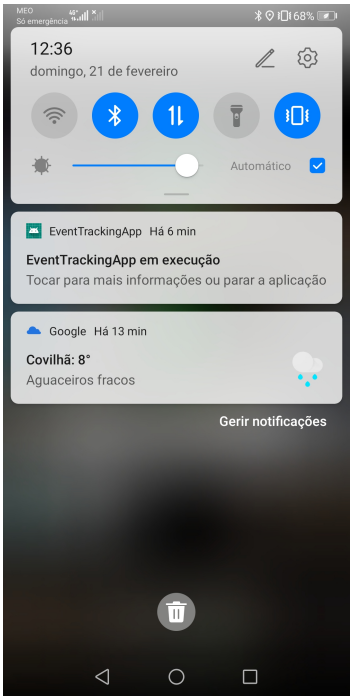

Figura 6.25: Serviço "Tracking Service"

Se fizermos uma pausa de pelo menos 20 minutos, o trajeto entre o momento da pausa e o momento em que voltamos a andar, vai ficar com cor vermelha isto para indicar que houve uma paragem de pelo menos 20 minutos (ver figuras 6.26 e 6.27)

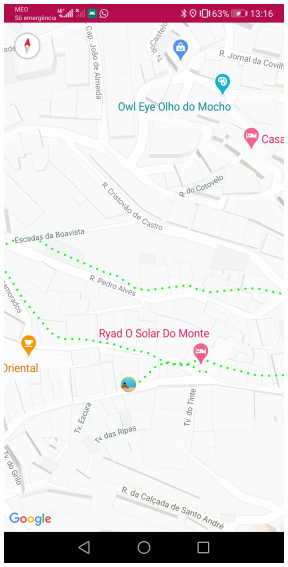

Figura 6.26: Momento da paragem

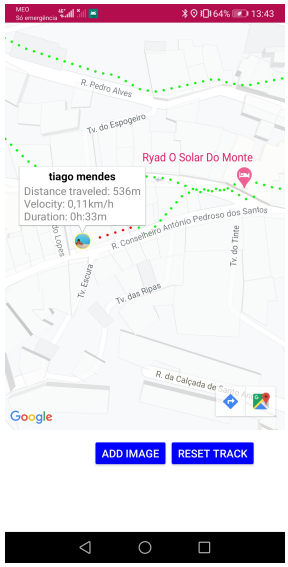

Figura 6.27: Momento depois da paragem

6.3 Testes

6.3.1 Testes Funcionais

Nesta secção são descritos os diversos testes efetuados à aplicação móvel.

Foram realizados diversos testes à aplicação com distintos *smartPhones*, com o objetivo de maximizar a deteção e depuração de diversos erros. Tendo alguns testes sido realizados com o utensílio de ferramentas de *debugging*. Estes testes podem ser consultados nas tabelas 6.1 e 6.2.

ID	Descrição do Teste	Resultado Esperado
0	Tentativa de login com a password errada.	Erro no login e notificar o utilizador.
1	Registo e as passwords diferem.	Erro no registo, o utilizador é notificado.
2	Registo em que falta um campo obrigatório	Erro no registo e notificar o utilizador.
3	Tentativa de escolha de uma password com menos de seis carateres.	Erro no Registo e notificar o utilizador.
4	Criação de Evento com campos obrigatórios em falta.	Erro e notificar o utilizador.
5	Fechar Evento.	Não deixar adicionar Imagens nem enviar mensagens.
6	Reiniciar trajeto	Reinicializar os dados (distância, velocidade) e posições do utilizador.
7	Mudar de localização.	Mostrar nova posição e o percurso até a nova posição.
8	Adicionar Imagem.	Mostrar imagem no mapa.
9	Clicar na própria marca no mapa	Mostrar distância e velocidade.
10	Estar num evento e ter aplicação em segundo plano	A aplicação deve continuar a guardar as posições novas e ir atualizando as posições do utilizador.
11	Registo com email já utilizado	Deve dar erro no registo

Tabela 6.1: Testes a realizar.

ID	Resultados Obtido aos testes da tabela 6.1
0	Mostra notificação “authentication Failed!”
1	Mostra notificação de erro “The passwords do not match!”
2	Mostra notificação de erro “Missing Username or Email!”
3	Mostra notificação de "Erro no Registo".
4	Mostra aviso dos campos em falta.
5	Como esperado não deixa adicionar imagens ao evento nem mensagens, apenas consultar o percurso, imagens existentes e mensagens trocadas.
6	A distância é reinicializada para 0 metros tal como a velocidade e o trajeto antes percorrido deixa de ser mostrado.
7	A marca do utilizador aparece na nova localização e é traçado o trajeto até a nova posição.
8	Um ícone da nova imagem é mostrado no mapa, e a nova imagem pode ser visualizada.
9	Permite ver a distância percorrida e a velocidade do utilizador.
10	As novas posições ficam guardadas no *Cloud Firestore* e ao voltar a aplicação este mostra todo o percurso até a nova localização.
11	Não completa o registo e mostra aviso "Register Failed"

Tabela 6.2: Resultado dos testes realizados.

6.3.2 Testes de performance

Para fazer testes de performance foi utilizado um serviço de monitoramento de desempenho do *Firebase*,[24]. Para isto é preciso adicionar umas dependências (excerto de código 6.1).

```
implementation platform ('com.google.firebase:firebase-bom:26.3.0')

implementation 'com.google.firebase:firebase-perf'
```

Listing 6.1: Excerto de código: Dependências para monitorizar performance

Com este serviço é possível monitorizar específicos trechos de código, medindo o tempo de certas funcionalidades. Implementar esta monitorização é bastante simples basta inicializar um *Trace* e atribuir um identificador a este e depois terminar este *Trace* posteriormente do código que queremos analisar.

Tendo sido esta funcionalidade utilizada para medir vários aspetos representados na tabela 6.3.

Trace ID	Descrição	Tempo	Amostras
_app_start	Tempo que a aplicação demora a iniciar	487ms	
full_T_create_user	Tempo total na criação de um utilizador	19,59s	13
creating_user_default_image	Tempo entre clicar no registar e ir para o dashboard, sem escolher imagem	1,07s	12
creating_user	Tempo entre clicar no registar e ir para o dashboard, com escolha de imagem	4,89s	10
open_chat_load_all_messages	Tempo ler todas as mensagens e poder interagir com chat	3,26s	7
full_T_add_image	Tempo total no fluxo de adicionar imagem	31,26s	5
adding_image	Tempo entre clicar guardar e voltar para o mapa	5s	5

Tabela 6.3: Traces criados e resultados.

Com estes Traces é possível tirar algumas conclusões como o tempo de criar um utilizador com a imagem default e com escolha de imagem aumenta drasticamente de 1,07s para 4,89s isto se deve a ter de fazer a atualização à *Firebase Storage* da nova imagem. O firebase também permite visualizar os tempos de pedidos de rede conseguindo ver o tempo que demora a fazer este pedido (mostrado na figura 6.28) sendo este de 785ms que é o pedido realizado à *firebaseStorage.googleapis* o outro pedido é quando se faz o login pelo provedor da *Google* e o Firebase vai buscar a informação do Utilizador ao *Google*.

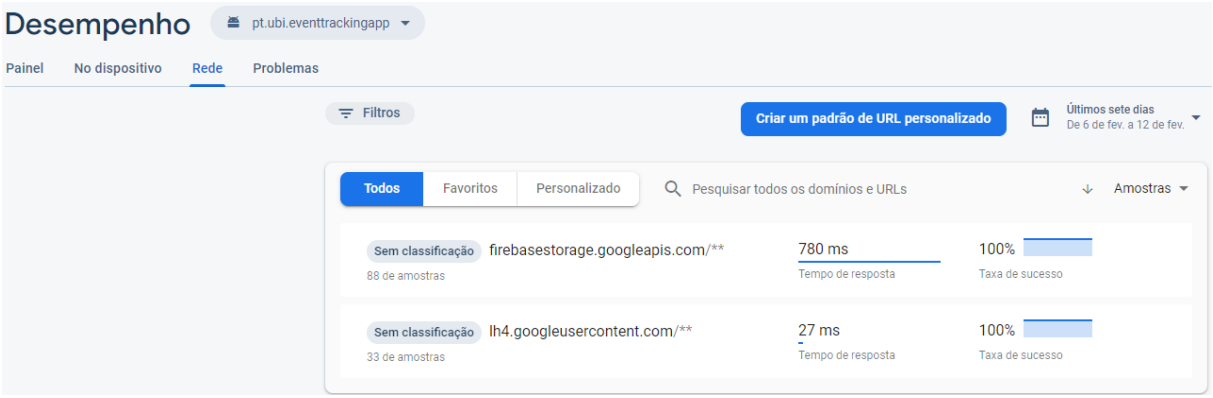

Figura 6.28: Pedidos de rede

6.3.3 Demonstração de alguns testes realizados

Ao longo da realização da aplicação foi necessário fazer vários testes, esta secção demonstra alguns testes realizados ao longo do desenvolvimento da mesma.

6.3.3.1 Primeiro teste realizado

Nesta prova foi testado a sala de conversa tendo sido realizada uma pequena conversa entre mim e outro utilizador que decorreu em Dezembro de 2020 (figura 6.31). Foi também testado a funcionalidade de adicionar imagens ao mapa, tal como mostrar o trajeto percorrido, figura 6.30 e 6.29, aqui foi reparado que devia ter duas cores diferentes para o percurso, com a finalidade de identificar uma determinada pausa no percurso, tendo sido definido a verde se o percurso tivesse sido percorrido em menos de 20 minutos e vermelho caso contrário. Foi também reparado que a aplicação estava a *crashar* algumas vezes, principalmente depois de escolher uma imagem para adicionar ao mapa, tendo depois isto sido resolvido substituindo o *Glide* pelo *Picasso*5.2.3.

Figura 6.29: Teste do mapa com 3 imagens

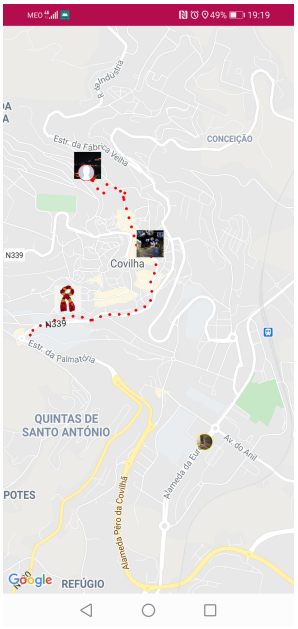

Figura 6.30: Teste do mapa com percurso

Figura 6.31: Teste da sala de conversa

6.3.3.2 Segundo teste realizado

Nesta prova já podemos ver informação ao clicar na marca do utilizador como distância percorrida e velocidade, como também vários problemas resolvidos como os *crash* que antes aconteciam (figuras 6.32 e 6.33) este teste foi realizado a 10 de

Fevereiro de 2021.

Figura 6.32: Adicionar Imagem ao mapa

Figura 6.33: Marca com distância e velocidade

6.3.4 Testes com nova implementação tendo em conta os resultados anteriores

Como podemos ver pelos testes de performance 6.3, adicionar uma imagem demora 5s, isto é muito tempo para o utilizador ficar a espera antes de voltar para o mapa, então foi implementado uma abordagem optimista, ou seja, em vez do pedido de adicionar a informação da imagem a *Firebase Cloud* e a imagem ao *Firebase Storage* serem pedidos *synchronous*, o utilizador espera pela resposta do *Firebase*, estes passaram a ser pedidos *asynchronous* não sendo preciso o utilizador esperar pela resposta do servidor. Nesta abordagem é preciso avisar o utilizador que a imagem esta a ser adicionada e depois se esta foi adicionada com sucesso ou não (figuras 6.35 e 6.36). Esta abordagem reduziu o tempo de espera de 5s para 59ms, como ilustrado na figura 6.34.

Figura 6.34: Tempo da abordagem optimista vs pessimista

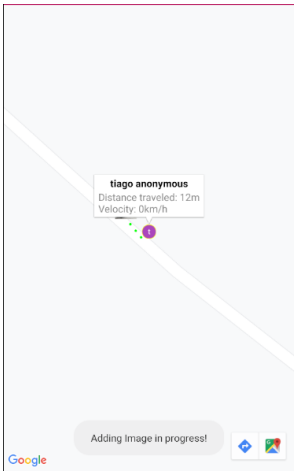

Figura 6.35: Avisar utilizador imagem a ser adicionada

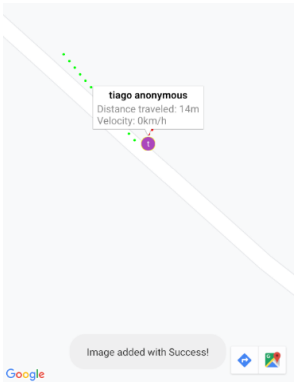

Figura 6.36: Avisar utilizador imagem adicionada

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Conclusões Principais

Este trabalho constitui uma concretização pessoal na área do desenvolvimento de aplicações Android com GPS.

A aplicação final cumpre com os objetivos principais definidos no inicio do seu desenvolvimento:

- Permite a criação de um evento que possa ser acedido por outros utilizadores;
- Permite visualizar o próprio percurso ao longo do decorrer de um evento;
- Permite visualizar alguns dados do percurso como distância e velocidade;
- Permite a partilha de imagens e visualização das mesmas no mapa;
- Permite ver a posição de outros utilizadores no mapa pertencentes ao mesmo Evento;
- Permite que os utilizadores comuniquem utilizando uma sala de conversa;
- Permite o encerramento de um evento bem como a sua eliminação;

Com este trabalho aprofundei o meu conhecimento em Android, principalmente sobre fragmentos e comunicação entre atividades e fragmentos. Um fragmentos define e gerência o seu próprio **layout**, tendo seu próprio ciclo de vida e pode lidar com os seus próprios eventos, sendo possível combinar vários fragmentos em uma única atividade.

Um fragmento tem de ser hospedado por uma atividade, sendo este impactado pelo ciclo de vida da atividade. A utilização de fragmentos foi uma das dificuldades enfrentadas ao longo da realização deste projeto, tendo começado inicialmente por ter desenvolvido uma atividade que continha dois fragmentos, mapa e sala de conversa, depois de algum tempo com esta implementação decidi mudar pois a sua complexidade aumentou gradualmente quando tive de adicionar também o fragmento de adicionar uma imagem e também o fragmento de rodapé, optando depois por utilizar duas atividades uma para o mapa e outra para a sala de conversa, sendo a atividade do mapa na verdade uma *FragmentActivity* visto que precisei de usar *nested fragments*.

Utilizei também um serviço para atualizar a localização de um utilizador em segundo plano. Aprofundei também os meus conhecimentos sobre a API da *Google Maps* e as funcionalidades que esta disponibiliza. Posso dizer que o desenvolvimento deste projeto me ajudou a perceber bem o desenvolvimento de aplicações móveis que recorrem à posição do utilizador, bem como, a perceber como utilizar uma base de dados na Cloud, *NOSQL*, como o *Firebase*, para o armazenamento de informação.

7.2 Trabalho Futuro

Consegui completar todos os objetivos deste trabalho apesar de poderem ser feitas algumas melhorias como:

- Ser possível mostrar mais informação sobre um percurso por exemplo mostrando um ecrâ com um gráfico apresentando diferentes velocidades dividindo o trajeto em várias secções, mostrar também dados como a elevação;
- Seria também interessante adicionar o tipo de evento, se é uma caminhada a pé, bicicleta ou de carro isto permitiria ajustar os tempos a que as posições do utilizador seriam guardadas, tempos definidos no *LocationTrackingService* (presente na subsecção 5.2.1.4) ;
- Criar uma aplicação *web* em que seja possível gerir os seus eventos e poder criar eventos, podendo também convidar pessoas para o evento;
- Seria também interessante poder adicionar uma password a um Evento e ter de ser necessário inserir uma password para ingressar no Evento;

Bibliografia

[1] Cuponation. Usuarios smartphones 2020, 2019. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/, último acesso: 29 Janeiro 2021.

[2] statista. Number of smartphone users worldwide, 2019. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/, último acesso: 29 Janeiro 2021.

[3] Statcounter. Os market share in mobile, 1999. https://gs.statcounter.com/os-market-share/mobile/worldwide, último acesso: 16 Janeiro 2021.

[4] Stephanie Rosenbloom. Where is everyone try a tracking app, 2013. https://www.nytimes.com/2013/06/16/travel/where-is-everyone-try-a-tracking-app.html, último acesso: 17 Janeiro 2021.

[5] TechTudo. Gps tracker pro. https://www.techtudo.com.br/tudo-sobre/gps-tracker-pro.html, último acesso: 16 Janeiro 2021.

[6] GeoTrack. Geotracker. https://geo-tracker.org/, último acesso: 16 Janeiro 2021.

[7] iSharingSoft. isharing, 2019. https://isharingsoft.com/, último acesso: 17 Janeiro 2021.

[8] Madhavan nagarajan. Sql vs. nosql databases: What's the difference?, 2019. https://medium.com/@itIsMadhavan/sql-vs-nosql-databases-whats-the-difference-a05492b48d99/, último acesso:24 Janeiro 2021.

[9] Jeffrey Jaffe e Tim Berners-Lee. W3c. https://www.w3.org/, último acesso:24 Janeiro 2021.

[10] Ibiblio. O que é sgml. http://www.ibiblio.org/godoy/sgml/docbook/porque/que-sgml.html, último acesso: 16 Janeiro 2021.

[11] Wang Yubin Huang Yi, He Qian. Research on gps positioning in mobile communication equipment based on android platform. 2014.

[12] John Kyes. what is gps, 2020. https://www.geotab.com/blog/what-is-gps/, último acesso: 30 Janeiro 2021.

[13] Joel McNamara. Gps for dummies. 2004.

[14] Learn Engineering. Gps, how does it work?, 2019. https://www.youtube.com/watch?v=8eTlI19_57g&ab_channel=LearnEngineeringde, último acesso: 15 Feveiro 2020.

[15] Google. Android studio, 2013. https://developer.android.com/studio, último acesso: 17 Janeiro 2021.

[16] Google. Firestore documentation. https://firebase.google.com/docs/firestore, último acesso:26 Janeiro 2021.

[17] Scott Carey. How spotify migrated everything from on-premise to google cloud platform, 2018. https://www.computerworld.com/article/3427799/how-spotify-migrated-everything-from-on-premise-to-google-cloud-platfo
html, último acesso:27 Janeiro 2021.

[18] Google. Firebase documentation, 2010. https://firebase.google.com/docs,
último acesso:24 Janeiro 2021.

[19] Google. Location request. https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest,
último acesso: 17 Feveiro 2020.

[20] Sam Judd. Glide git hub. https://github.com/bumptech/glide,
último acesso: 14 Feveiro 2020.

[21] Sam Judd. Glide v4 fast and efficient image loading for android. https://bumptech.github.io/glide/,
último acesso: 14 Feveiro 2020.

[22] Brian Geary. Fresco vs picasso vs glide, 2018. https://www.andplus.com/blog/fresco-vs-picasso-vs-glide,
último acesso: 14 Feveiro 2020.

[23] Shiva singh. Difference between picasso and glide, 2018. https://medium.com/@singhsiva177/difference-between-picasso-and-glide-5d4b944c7088,
último acesso: 14 Feveiro 2020.

[24] Google. Firebase performance documentation, 2010. https://firebase.google.com/docs/perf-mon?authuser=1,
último acesso: 12 Feveiro 2020.

.