Instituto Superior de Engenharia de Lisboa
Área Departamental de Engenharia Eletrónica e Telecomunicações e de Computadores


# Aplicação para reportar ruído excessivo em zonas urbanas

Alexandre Ruivo
42190, a42190@alunos.isel.pt, 938210475
Filipe Alexandre
42210, 42210@alunos.isel.ipl.pt, 918816646
Rafael Batalha
42188, 42188@alunos.isel.ipl.pt, 910400737

**Licenciatura em Engenharia Informática e de Computadores**

Relatório de Progresso

**Orientadores:**
José Simão, jsimao@cc.isel.ipl.pt
Nuno Datia, datia@isel.ipl.pt

Abril de 2019

# Índice

# Introdução

## 1.1 Enquadramento do problema

Em zonas de diversão noturna, é frequente que após o fecho dos estabelecimentos comerciais os clientes se mantenham nas imediações dos ditos estabelecimentos, provocando ruído indesejado aos moradores das zonas. Por norma, as forças da lei são avisadas mas encontram dificuldades em entender se o ruído era excessivo dado que apenas conseguem aferir o que se passa no momento em que chegam ao local e não quando o morador reporta às autoridades.

## 1.2 Objetivos

Este projeto visa dotar o cidadão um meio para reportar, através de gravações, o ruído existente na sua zona de residência, permitindo às forças da lei visualizar as áreas com maior quantidade de queixas que contenham ruído excessivo e que assim apliquem uma distribuição dos seus meios de forma mais eficiente e preventiva.
Também é objetivo desta solução fornecer uma interface de utilização para dar acesso tanto a funcionalidades de utilizador, como registar uma queixa, como funcionalidades de gestão da informação, como obter todas as queixas para uma determinada área.

## 1.3 Paralelos com o mundo real

Após a identificação do problema, foi encontrada uma aplicação que procura solucionar um problema semelhante ao descrito denominada de *Na Minha Rua LX* (1). Esta aplicação permite aos utilizadores reportar problemas de múltiplos tipos (e.g. árvores caídas, estacionamento abusivo) ocorridos na cidade de Lisboa e não especificamente problemas de ruído excessivo. No entanto, é possível desenhar paralelos com esta aplicação, nomeadamente no que toca à criação e processamento de queixas. Para além do paralelo descrito, é também possível observar outra semelhança no que toca à visualização das queixas, principalmente em relação a dispo-las num mapa e verificar quais as áreas com maior número de queixas.

## 1.4 Proposta de Solução

A proposta de solução vai assim ser constituída por uma aplicação *Android*, por uma aplicação Web e por um componente servidor.

A aplicação *Android* vai dar capacidade aos utilizadores de se registarem e de puderem submeter queixas se o ruído detetado for considerado excessivo. A aplicação Web permite a visualização das queixas de forma geo-referenciada enquanto que o componente servidor disponibiliza uma API para interação com ambas as aplicações.

# Arquitetura da Solução

O projeto, de uma forma resumida, vai estar dividido em seis componentes:

- Uma base de dados;
- Uma API;
- Uma aplicação móvel;
- Uma aplicação web.

Ambas as aplicações comunicam com o servidor através da API que o mesmo exporta. A figura 1 ilustra a organização dos diferentes componentes do projeto. A figura 2.1 demonstra como interagem os diferentes componentes do projeto.

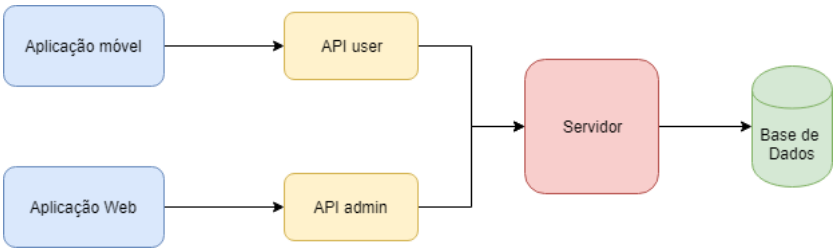


Figura 2.1: Organização dos componentes do projeto

## 2.1 Base de dados

A base de dados a ser utilizada é *PostgreSQL* (2), *open source object-relational* que, através do plugin *PostGIS* (3), permite a utilização de funcionalidades geo-referenciadas.

## 2.2 API

A API permite que os utilizadores se registem na aplicação e que submetam queixas quando considerarem que exista ruído excessivo. Utiliza *JSON* (4) para representação de recursos e *Problem+JSON* (5) para representação de erros. Esta permite aos utilizadores registados como operadores visualizarem as queixas de todos os utilizadores, disponibilizando também um conjunto de filtros espaciais e temporais.Adicionalmente, permite que o operador observe detalhes mais profundos sobre a queixa e dados sobre os utilizadores que reportem as ditas queixas.

## 2.3 Aplicação Móvel

A aplicação móvel tem como finalidade disponibilizar uma interface gráfica aos utilizadores para o registo dos mesmos e da submissão de queixas. Permite também que o utilizador possa

consultar o estado da sua queixa (i.e. se algum operador já teve em consideração a dita queixa).

A aplicação móvel é responsável por efetuar o processamento de sinal necessário para determinar se o áudio capturado corresponde a ruído excessivo ou não.

A plataforma escolhida para o desenvolvimento da aplicação móvel foi *Android*, tendo em consideração o conteúdo lecionado no ciclo de estudos (Unidade Curricular de *Programação em Dispositivos Móveis*) e também o facto de nenhum dos elementos do grupo ter acesso fácil a um sistema *macOS*, necessário para compilação de aplicações para *iOS*.

## 2.4 Aplicação Web

A aplicação web corresponde ao componente que permite aos operadores a visualização de queixas, permitindo a filtragem por datas, horas e por localização. Também permite a visualização das queixas sob a forma de um mapa de calor, podendo-se assim observar com maior facilidade as áreas mais problemáticas da cidade. Disponibiliza também a visualização de detalhes sobre a queixa como, por exemplo, os níveis de decibéis medidos na gravação original, e também detalhes sobre o utilizador que submeteu a queixa. Irá ter também uma página com estatísticas gerais sobre todas as queixas submetidas até à data.

# Implementação

## 3.1 Base de Dados

A escolha da base de dados é um passo importante no desenvolvimento do projeto pois dependendo da base de dados escolhida, a forma como os dados são acedidos vai alterar-se, o que constitui uma parcela considerável na implementação da componente de servidor.

### 3.1.1 Relacional ou *NoSql*?

A decisão entre utilizar um sistema baseado no modelo relacional ou num modelo *NoSql* acaba por ser determinada pelo requisitos da aplicação a desenvolver. No caso deste projeto em concreto, é necessário ter em consideração que, no fundo, o que está a ser desenvolvido pode ser considerado um *Geographic information system* (6) ou, de forma abreviada, um *GIS*, visto que se está a indexar, com base em coordenadas geográficas, objetos que mapeiam um dado domínio, que neste caso corresponde ao do ruído excessivo em zonas de diversão noturna.

Tendo em consideração a premissa anteriormente mencionada, um artigo (7) que compara as opções disponíveis para bases de dados com funcionalidades geo-espaciais relacionais e *NoSql*, as considerações dos orientadores, e que se pretendia uma ferramenta *open source*, ficou decidido que o sistema a utilizar seria uma base de dados relacional *PostgreSQL* com o *plugin PostGIS*.

### 3.1.2 *PostGIS*

*PostGIS* é um *plugin* para *PostgreSQL* que permite a utilização de tipos de dados geográficos, interrogações com base nesses pontos geográficos e em parâmetros como, por exemplo, um raio sobre um dado ponto. Permite ainda a utilização de funções sobre esses tipos espaciais.

### 3.1.3 Modelo de Dados

No contexto da aplicação, um participante pode corresponder tanto a um operador como a um utilizador dito 'comum'. O *role* distingue os tipos de utilizadores, e assim, as permissões a que o utilizador tem acesso. Um utilizador pode estar associado a múltiplas queixas . A necessidade de guardar o identificador do dispositivo *Android* surge numa forma de tentar associar um utilizador malicioso (que envie queixas indevidas) a um utilizador de forma a que exista mais de uma forma de identificar o utilizador.

## 3.2 Servidor Aplicacional

### 3.2.1 Funcionalidades

A aplicação visa permitir a introdução de novos utilizadores e o *login* dos mesmos. Será ainda possível para um utilizador a criação de queixas e de obter todas as suas queixas, ou uma em especial. Estas queixas guardam coordenadas, através do tipo geográfico Ponto, disponibilizado

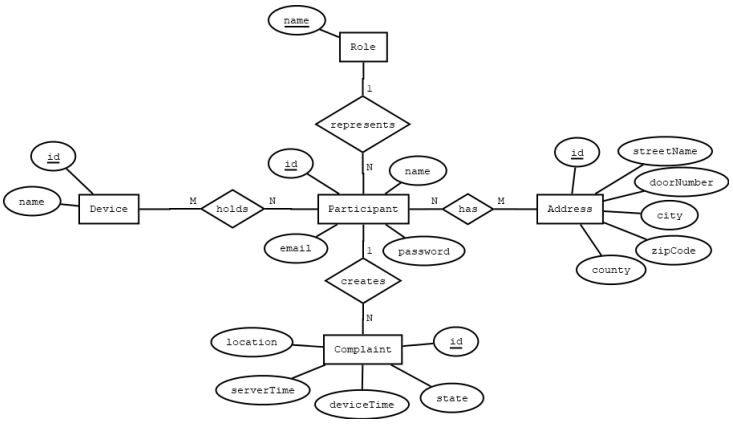


Figura 3.1: Modelo Entidade-Associação

pela extensão em cima descrita. Existe ainda uma lógica de *roles*. Cada utilizador pode ter três papéis diferentes:

- utilizador - Qualquer civil que precise de efetuar alguma queixa devido a barulho excessivo;
- *operador* - Entidade responsável por tratar as queixas e por analisar as estatísticas existentes sobre o assunto;
- *god* - Entidade com acesso total a todas as funcionalidades da aplicação.

Toda esta lógica de filtrar quem tem acesso e quem não tem é implementada à custa de filtros, que são implementados com o tipo concreto *FilterChainProxy*. Este tipo engloba todos os filtros gerados pelo *spring security*, onde está implementada toda a lógica de segurança, e apenas gera um único filtro na cadeia de filtros entre cliente e *Servlet*. Esse mesmo filtro delega, posteriormente, o seu trabalho à cadeia interna com o tipo em cima referido.

### 3.2.2 Plataformas possíveis

Foram identificadas as seguintes plataformas para a realização da componente servidor: *Spring (8), Ktor (9) e Express (10)*. A nossa decisão inicial dividiu-se entre *Spring* e *Express* sendo que foi decidido utilizar a tecnologia lecionada na unidade curricular DAW (Desenvolvimento de Aplicações Web), *Spring*.

A descoberta de *Ktor* ocorreu demasiado tarde para poder competir com a escolha previamente referida. Verificou-se que é uma plataforma expressiva no mundo *Kotlin* e que a sua utilização é bastante parecida com a de *Express*. Porém, a sua descoberta aconteceu depois de se ter iniciado o estudo de *Spring*, levando a que se mantivesse *Spring* como a plataforma escolhida.

### 3.2.3 *Spring*

*Spring Boot* é uma *framework* bastante personalizável, sendo possível substituir grande parte das peças que a compõem por implementações que o programador queira utilizar. Possui um mecanismo de injeção de dependências através da criação de *Beans*, sendo essa a principal razão pela qual é bastante personalizável.

É também uma plataforma muito utilizada, possuindo um elevado número de bibliotecas que

lhe podem ser adicionadas e integração com diferentes *frameworks*. Uma das *frameworks* que mais se destacou foi a que permite criar uma abstração sobre a camada de dados, implementando a especificação JPA: *Hibernate* (11). Dado ser uma plataforma bastante difundida também permite fornecer ao programador bastantes exemplos com diferentes variantes, facilitando a resolução de problemas quando estes aparecem.

### 3.2.4 Segurança

Tendo em conta que se trata de utilizadores, e utilizadores têm sempre informação sensível guardada no seu perfil, é necessária a implementação de alguma forma de segurança. Com base neste problema, após alguma pesquisa, identificámos três possíveis mecanismos de autenticação: *Http Basic* (12), *OpenIdConnect* (13) e *JWT* (14) para guardar as credenciais de um utilizador. Embora a nossa escolha tenha recaído sobre a última opção, de momento está implementado *Basic Auth* apenas como algo experimental, de modo a ter algum meio de autenticação.

#### 3.2.4.1 *Http Basic* vs *OpenIdConnect* vs *JWT*

A primeira possibilidade, *Http Basic*, foi afastada desde cedo porque implica que o conteúdo do *header Authentication* passe pelo canal de comunicação codificado como *Base64*. Como tal, mesmo sabendo que a comunicação ocorre através de um canal seguro, um utilizador malicioso que intercete a comunicação é capaz de descodificar a informação, obtendo assim acesso às credenciais do utilizador. Visto que não consideramos possível a autenticação através de quaisquer outras entidades, com a finalidade de evitar, ao máximo, contas falsas e maus utilizadores, *OpenIdConnect* teve de ser também excluído da lista de hipóteses. A opção do *JSON Web Token* resolve em simultâneo a questão da autenticação e da autorização, visto que contém informação de autenticação e *claims* sobre o utilizador, *claims* estas que definem o seu papel na aplicação, toda esta informação está guardada num objeto json que é então assinado com um esquema de cifra assimétrica, mais propriamente assinatura digital, garantindo assim a integridade dos dados. Pode-se concluir então que a autenticação é sempre feita diretamente na aplicação.

#### 3.2.4.2 Dificuldades

Até ao momento, as dificuldades passaram essencialmente por perceber a configuração do *spring-security*, e assim que essa dificuldade foi ultrapassada, não foi muito difícil compreender como seriam implementados os filtros subsequentes aos pedidos à API.

## 3.3 Aplicação Móvel

A aplicação *Android* permite ao utilizador reportar ruído, consultar todas as suas queixas, e receber notificações para quando estas mudem de estado. Também é possível alterar alguns dos seus dados pessoais, ou até mesmo a descrição de uma queixa que acabou de fazer.

### 3.3.1 Arquitetura MVVM

A arquitetura *Model-View-ViewModel* (15) tem como principal objetivo a separação dos componentes de utilização da interface gráfica, camada de negócio e do modelo de armazenamento. Permite também uma melhor testabilidade de cada um destes componentes. Este

padrão de desenho é sugerido pelo Android Jetpack (16) para o desenvolvimento de aplicações *Android*. A figura seguinte demonstra esse mesmo padrão.

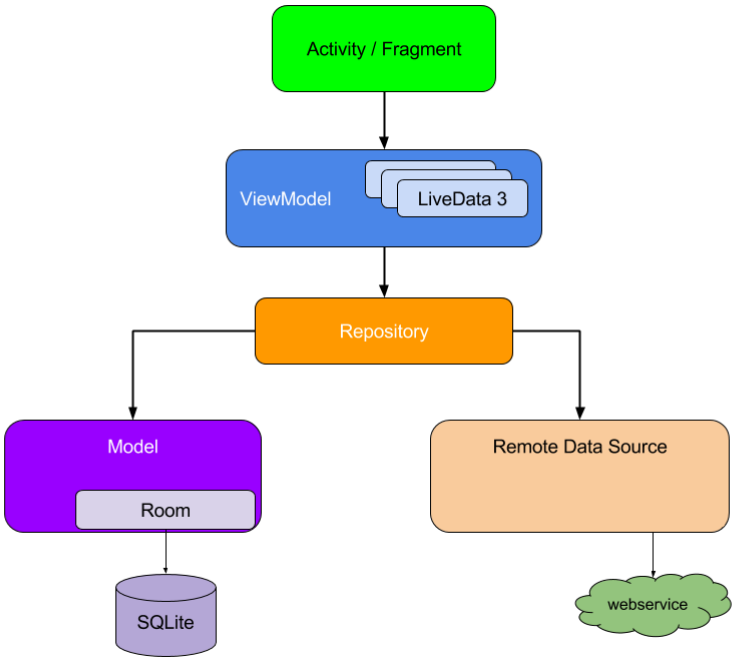


Figura 3.2: Repositório sugerido pelo Android Jetpack. Figura adaptada da imagem presente em (17)

O *Model* representa a camada onde os dados são guardados, seja localmente na aplicação, ou no servidor. É no *Repository* que é feita toda a lógica de armazenar os dados, assim como as decisões de enviar para o servidor ou guardar localmente.

A *View* é constituída pelos componentes da interface gráfica, mostrando os dados ao utilizador. De forma a que esta se encontre sempre atualizada, os dados são sempre observados quando existem mudanças de estado.

No *ViewModel* são persistidos os dados correspondentes a cada *View*. Desta forma, uma *View* tem tantos *ViewModel* quanto os diferentes dados diferentes que esta apresente ao utilizador.

Visto que as *View* observam os seus *ViewModel* para estarem sincronizadas com os dados, isto é conseguido com a utilização do componente *LiveData*.

A utilização deste componente é fundamental, pois assim garantimos que toda a interface apresentada ao utilizador corresponde à informação armazenada. Não existe qualquer *memory leak* pois todos estes objetos existem enquanto a *Activity* existir (tem a noção do seu ciclo de vida). Caso haja alguma alteração na configuração da aplicação, por exemplo, devido à rotação do ecrã, a informação mostrada é a mais recente.

### 3.3.2 Consistência dos dados

Sendo que os dados enviados pelo servidor apresentam o formato *JSON*, é necessário que estes sejam devidamente convertidos para objetos modelo.

De modo a evitar os pedidos entre a aplicação e o servidor sobre informação que não foi


Figura 3.3: *Activity* de login

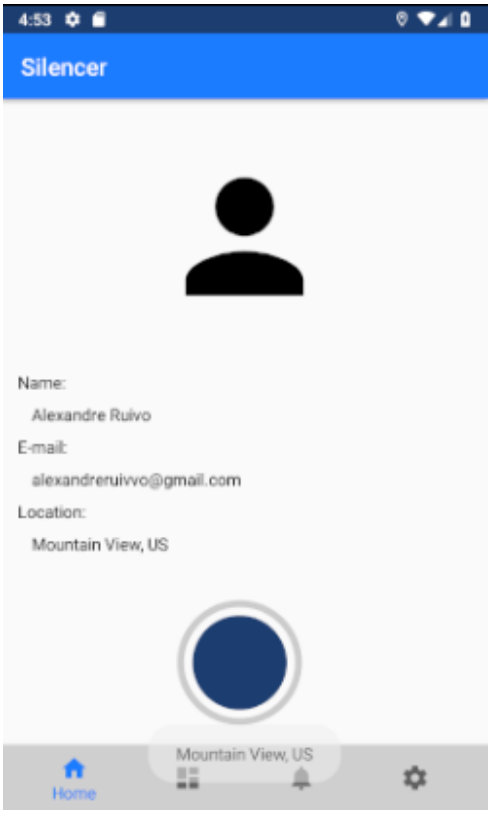


Figura 3.4: *Activity* de ecrã principal

alterada, essa informação é armazenada localmente numa base de dados através de uma camada de abstração fornecida pelo componente *Room* (18), onde não é necessário que haja preocupação com a serialização/desserialização da informação pretendida. Desta forma, as interrogações à base de dados são feitas através de anotações.

## 3.4 Aplicação Web

A aplicação web permite a um operador consultar as queixas registadas, podendo filtrar por data, horas do dia e por localização (fazendo uso do zoom corrente do mapa). A aplicação também disponibiliza a possibilidade de visualizar as queixas sob a forma de um mapa de calor, podendo-se observar com maior facilidade as zonas da cidade que requeiram mais atenção por parte das autoridades. Também é possível obter detalhes concretos sobre uma queixa e sobre o utilizador que submeteu a queixa. A escolha da *framework* a utilizar para a *user interface* recaiu sobre *React* (19) dado que é utilizada como objeto de aprendizagem no ciclo de estudos.

### 3.4.1 Arquitetura

Uma aplicação *React* é feita à custa de componentes. Estes componentes podem ser *stateless* ou *stateful*. Componentes *stateless*, como o nome indica, não guardam estado. Estes limitam-se a indicar como um dado elemento será exibido ao utilizador. Um componente *stateful* mantém estado, ou seja, guarda dados que são dinâmicos, alterando o seu aspeto conforme os dados dos quais vai mantendo registo. No contexto desta aplicação, por exemplo, um componente *stateful*

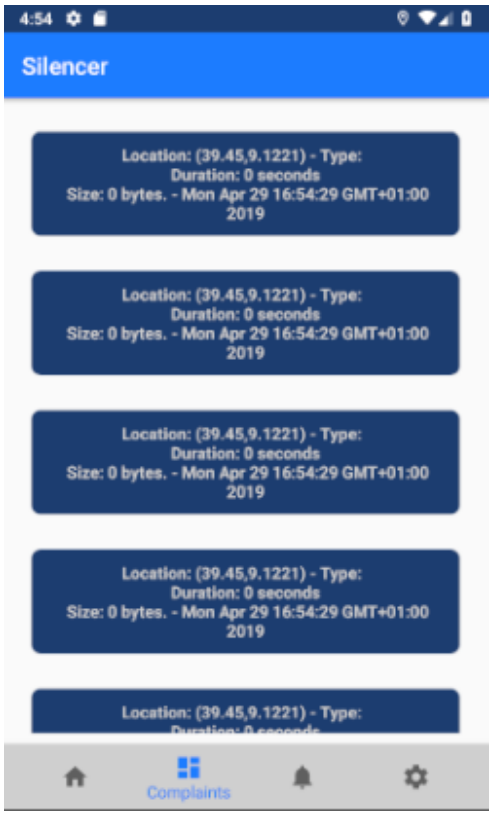


Figura 3.5: *Activity* com a lista de queixas de um utilizador


Figura 3.6: Área de notificações

pode ser o contentor do mapa, que vai alterar as queixas que exibe conforme os filtros aplicados, enquanto que em componente *stateless* pode ser, por exemplo, o que exibe os dados detalhados de uma queixa, dado que estes não se alteram ao longo do tempo.

A representação dos dados geográficos está a ser realizada com a biblioteca*leaflet* (20). Como a aplicação está a ser desenvolvida em *React*, utiliza-se a abstração da biblioteca *leaflet* em componentes *React*, denominada de *React-Leaflet* (21). O mapa de calor é implementado à custa da biblioteca*react-leaflet-heatmap-layer* (22).

### 3.4.2 Funcionalidades

Neste momento, é possível consultar os dados resumidos de uma queixa através de *popups* no mapa, e também de visualizar as queixas por meio do mapa de calor. Também é possível já consultar uma página de estatísticas gerais da aplicação e uma página de *login*, embora estas ainda não apresentem as funcionalidades pretendidas. Note-se que a aplicação, neste momento, ainda não faz pedidos ao servidor, sendo os dados das queixas *mocks* guardados em memória quando a aplicação arranca. O próximo passo consiste em começar a ligar a aplicação ao servidor e a introduzir autenticação na aplicação.

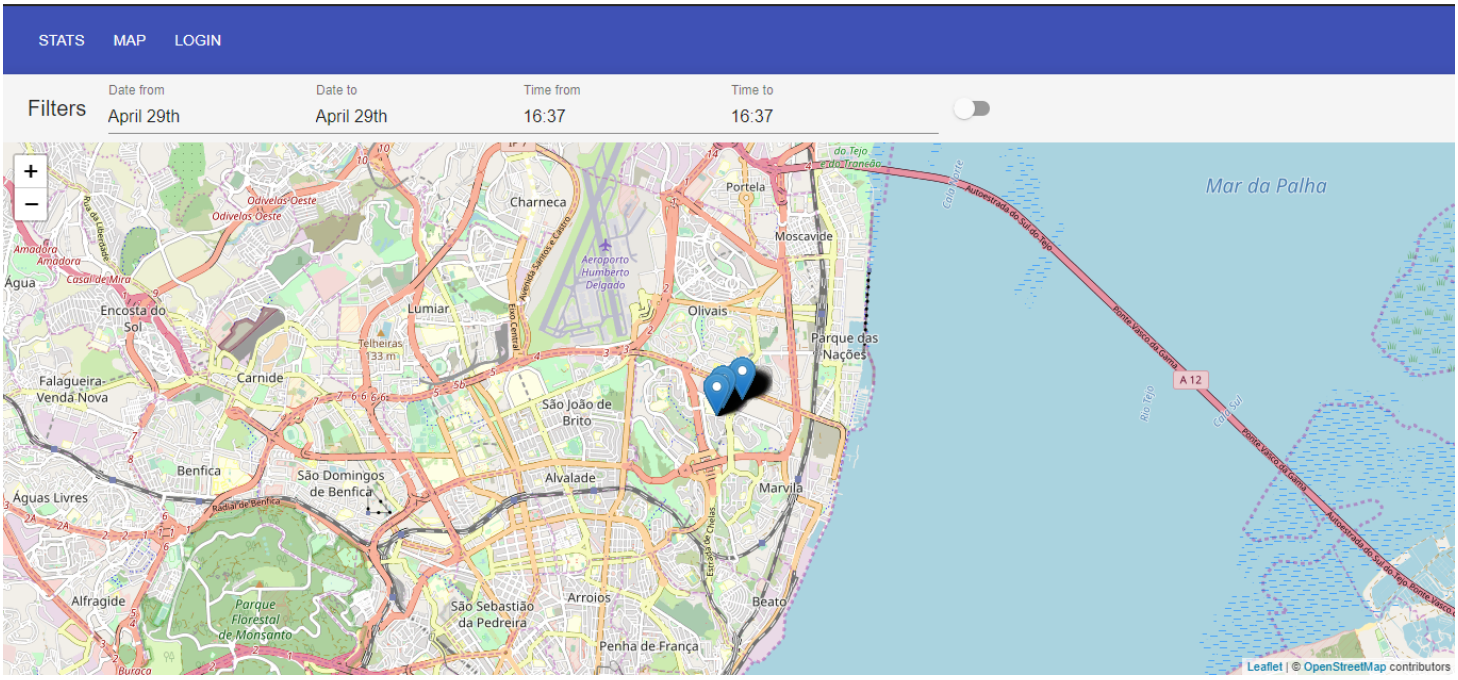


Figura 3.7: Demonstração do mapa com marcadores

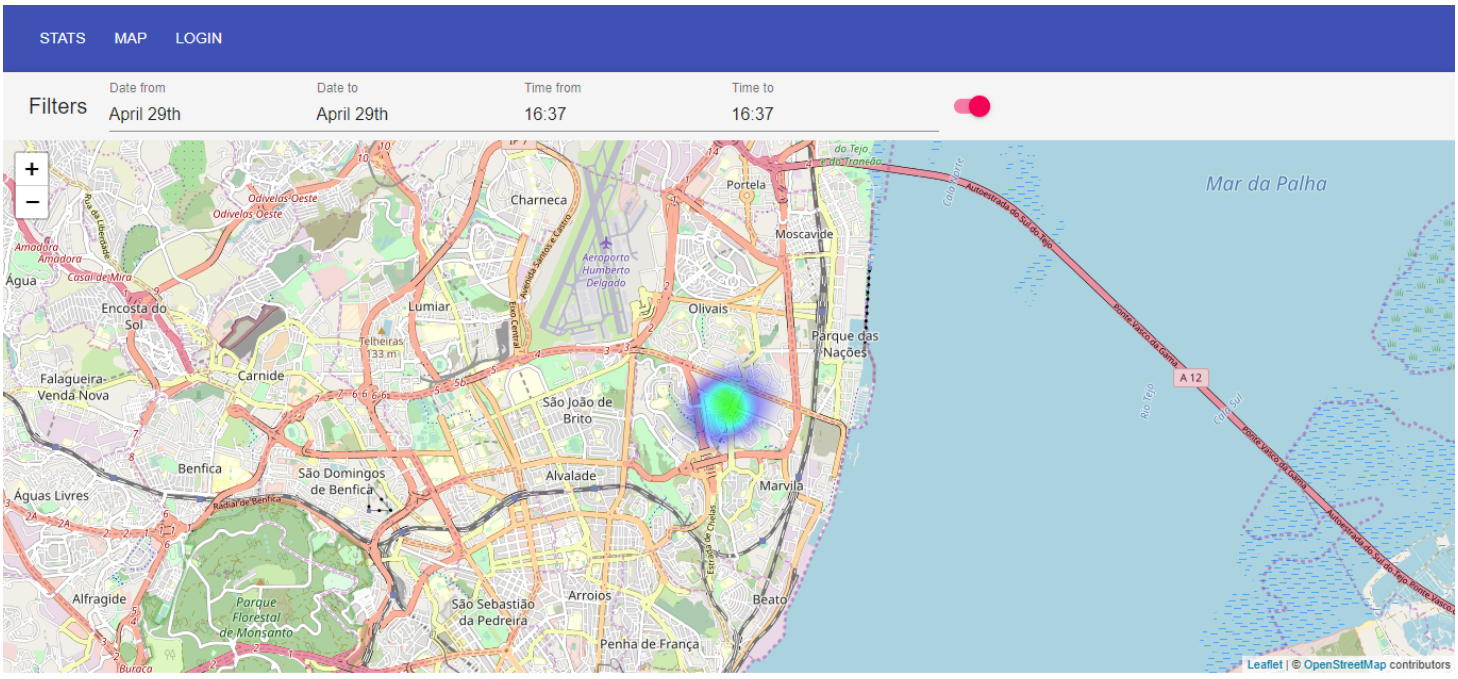


Figura 3.8: Demonstração do mapa de calor

# Conclusão

De uma forma geral, o progresso do projeto está de acordo com o planeamento apresentado anteriormente, na proposta de projeto, e novamente, na figura seguinte. Numa fase inicial, devido à pesquisa e aprendizagem de algumas tecnologias novas, o desenvolvimento ocorreu de forma mais lenta. No entanto, após ultrapassado esse obstáculo, o ritmo de trabalho aumentou de forma considerável.

As aplicações cliente estão numa fase em que têm apenas implementadas as funcionalidades básicas, nomeadamente recolha de áudio, na aplicação móvel, e obtenção de marcadores para colocar no mapa, ao nível da aplicação web. Em relação ao servidor, este já tem a camada de acesso a dados e autenticação implementadas. O consumo e manipulação de dados vai ser adicionado brevemente às aplicações cliente. Na aplicação web, o fornecimento de dados estatísticos será realizado depois das funcionalidades essenciais estarem concluídas.

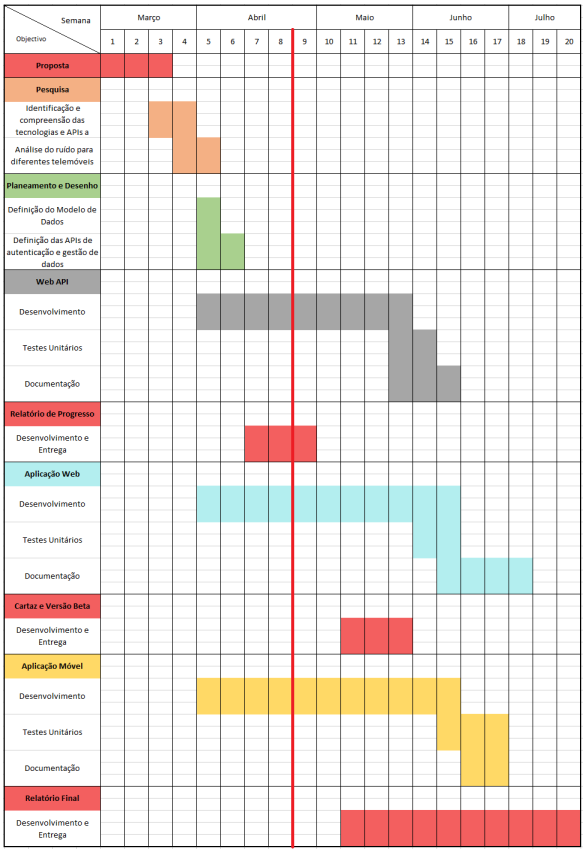


Figura 4.1: Planeamento apresentado inicialmente. A linha a vermelho demarca a semana atual.

# Bibliografia


[1] Câmara Munincipal de Lisboa. Na minha rua lx, 2019. URL https://naminharualx.cm-lisboa.pt/. [Online; acedido a 28-Abril-2019].

[2] The PostgreSQL Global Development Group. Postgresql, 2019. URL https://www.postgresql.org/. [Online; acedido a 28-Abril-2019].

[3] PostGIS Project. Postgregis, 2019. URL https://postgis.net/. [Online; acedido a 28-Abril-2019].

[4] Douglas Crockford. The javascript object notation (json) data interchange format, 2017. URL https://tools.ietf.org/html/rfc8259. [Online; acedido a 28-Abril-2019].

[5] Mark Nottingham. Problem details for http apis, 2016. URL https://tools.ietf.org/html/rfc7807#section-3. [Online; acedido a 28-Abril-2019].

[6] Wikipedia contributors. Geographic information system, 2019. URL https://en.wikipedia.org/wiki/Geographic_information_system. [Online; acedido a 28-Abril-2019].

[7] Baralis, Elena; Dalla Valle, Andrea; Garza, Paolo; Rossi, Claudio; Scullino, Francesco. Sql versus nosql databases for geospatial applications, 2017. URL https://zenodo.org/record/1149082/files/18_Baralis_et_al_2017_BSD.pdf. [Online; acedido a 22-Abril-2019].

[8] Pivotal Software. Spring framework 5, 2019. URL https://spring.io/. [Online; acedido a 29-Abril-2019].

[9] JetBreains. Ktor, 2019. URL https://ktor.io/. [Online; acedido a 29-Abril-2019].

[10] StrongLoop, IBM, and other expressjs.com contributors. Problem details for http apis, 2017. URL https://expressjs.com/. [Online; acedido a 29-Abril-2019].

[11] Red Hat Developers. Hibernate, 2019. URL https://hibernate.org/. [Online; acedido a 29-Abril-2019].

[12] Julian F. Reschke. The 'basic' http authentication scheme, 2019. URL https://tools.ietf.org/html/rfc7617. [Online; acedido a 29-Abril-2019].

[13] OpenID. Openid connect, 2019. URL https://openid.net/connect/. [Online; acedido a 29-Abril-2019].

[14] Michael B. Jones, John Bradley, Nat Sakimura. Json web token (jwt), 2015. URL https://tools.ietf.org/html/rfc7519. [Online; acedido a 29-Abril-2019].

[15] Wikipedia contributors. Model–view–viewmodel, 2019. URL https://en.wikipedia.org/wiki/Model\OT1\textendashview\OT1\textendashviewmodel. [Online; acedido a 29-Abril-2019].

[16] Android Developers. Android jetpack, 2019. URL `https://developer.android.com/jetpack`. [Online; acedido a 29-Abril-2019].

[17] Android Developers. Guide to app architecture, 2019. URL `https://developer.android.com/jetpack/docs/guide`. [Online; acedido a 29-Abril-2019].

[18] Android Developers. Room persistence library, 2019. URL `https://developer.android.com/topic/libraries/architecture/room`. [Online; acedido a 29-Abril-2019].

[19] Facebook Inc. React - a javascript library for building user interfaces, 2019. URL `https://reactjs.org/`. [Online; acedido a 29-Abril-2019].

[20] Vladimir Agafonkin. Leaflet, 2017. URL `https://leafletjs.com/`. [Online; acedido a 29-Abril-2019].

[21] Paul Le Cam and contributors. React-leaflet, 2019. URL `https://react-leaflet.js.org/`. [Online; acedido a 29-Abril-2019].

[22] OpenGov. react-leaflet-heatmap-layer, 2018. URL `https://github.com/OpenGov/react-leaflet-heatmap-layer#readme`. [Online; acedido a 29-Abril-2019].