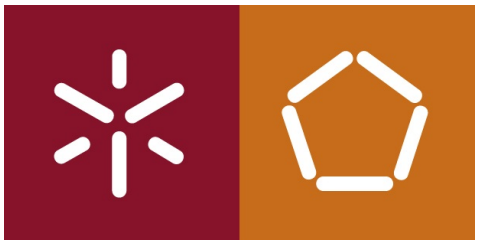

Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Engenharia Web

# Microservices

*Autores :*
Cesar Borges (A81644)
Fábio Senra (A82108)
Miguel Solans (PG41841)

11 de Junho de 2020

# Conteúdo

# 1. Introdução

Diariamente surgem novidades no mundo da informática e, cada vez mais o tema da condução autónoma está na ordem do dia. Futuramente, a condução autónoma trará vários benefícios como reduzir drasticamente o número de acidentes com veículos e por consequência melhorar a segurança nas estradas. O facto de reduzir o número de congestionamentos do trânsito irá diminuir a emissão de gases poluentes. Esta nova forma de mobilidade pretende ainda aumentar o acesso aos transportes de pessoas com mobilidade reduzida.

Este é o ponto de partida para o seguinte trabalho prático onde o principal objetivo deste é a construção de um sistema de monitorização de passadeiras para que os carros autónomos saibam se é seguro passar ou não numa determinada passadeira.

## 1.1 Estrutura do relatório

Inicialmente, o presente relatório incidirá sobre as tecnologias utilizadas em cada serviço, sejam estas bases de dados ou servidores da aplicaçao.

O capítulo Microservices incidirá exclusivamente sobre a arquitetura da Solução desenvolvida. Será dada a conhecer uma arquitetura generalizada de todo o sistema, apresentando o fluxo de pedidos e troca de informação entre micro-serviços. De forma a aprofundar a forma como cada micro-serviço foi implementado, serão apresentadas e discutidas as arquiteturas de cada serviço de forma pormenorizada.

De seguida, será apresentado a aplicação Cliente, recorrendo a diversas imagens, servindo também como manual de utilização da mesma.

Em forma de desfecho, será feita uma apreciação global sobre todo o trabalho desenvolvido.

# 2. Ferramentas

Nesta secção serão expostas todas as ferramentas utilizadas no desenvolvimento do sistema.

## 2.1 NodeJS

O ambiente de desenvolvimento da aplicação escolhido foi o nodeJS por ser um dos mais usados a nível mundial (comunidade grande e ativa) e por isso existe bastante suporte e apoio a sua utilização e principalmente por:

- a criação de aplicações é feita rapidamente
- o código é escrito com recurso à linguagem JavaScript que é dos que possui uma maior velocidade na execução do código (elevada performance)
- aceita muitas conexões simultâneas
- facilmente escalável (por exemplo, horizontalmente através da criação de nodos adicionais no sistema)
- NPM sempre em crescimento (Node Package Manager, fornece ferramentas e módulos para usar aumentando a produtividade de quem desevolve)

## 2.2 MySQL

A base de dados escolhida é o MySQL. As principais razões da sua escolha foram:

- Segura e confiável (usada por várias entidades como Facebook, Twitter)
- Open-source
- Das mais usadas e por isso com uma grande comunidade que fornece apoio

A utilização desta ferramenta está ser feita através do Sequelize, que é um nodeJS ORM baseado em promessas para várias bases de dados.

## 2.3 Redis

O Redis é um armazenamento de uma estrutura de dados em memória que é utilizada muitas vezes como cache, base de dados, entre outros. Foi decidido utilizar esta ferramenta uma vez que um dos micro-serviços não precisava de guardar muita informação e esta era volátil, ou seja, mudava com facilidade e rapidamente.

## 2.4 React

Para o frontend da aplicação foi utilizada a biblioteca React, sendo esta também uma das mais utilizadas do momento possui uma comunidade grande de suporte. Algumas outras vantagens são:

- Componentes reutilizáveis que poupam trabalho futuro
- Fácil manutenção devido à sua divisão
- Rápida aprendizagem

# 3. Microservices

Nesta secção serão apresentados todos os Microserviços que pretencem ao Sistema, com uma breve descrição e os diagramas de sequência considerados mais relevantes.

## 3.1. Arquitetura

O Sistema a desenvolver será decomposto em vários Microserviços, sendo que cada um representará diferentes papeis no contexto do Sistema.

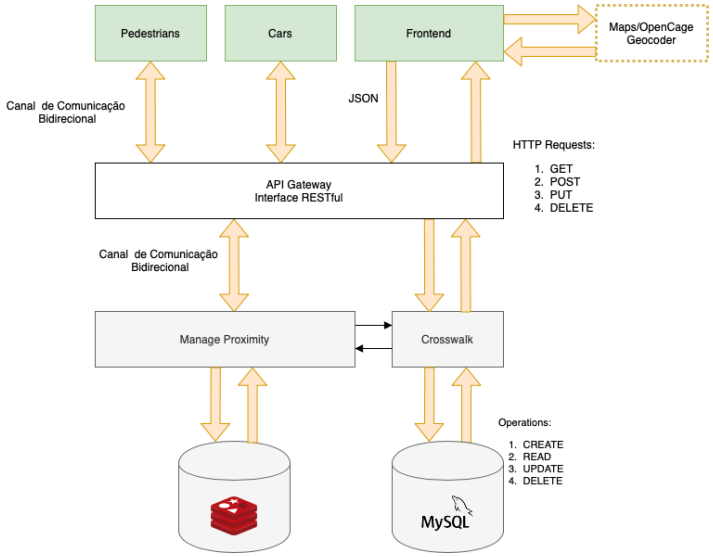


Figura 1: Arquitectura de Microserviços

Tal como pode ser observado na Figura 1, será utilizado um serviço de Base de Dados SQL, o MySQL, para armazenar informações sobre as diferentes Passadeiras. Estes dados deverão ser introduzidos pelos administradores do Sistema, após a sua autenticação, recorrendo a uma API de Dados - API Gateway - com uma Interface REST.

De forma a poder ser possível a interação e a visualização de dados com o utilizador final, será desenvolvido uma aplicação Frontend em React, de forma a consumir dados provenientes da API de Dados. Estes dados serão cruzados com um serviço de Mapas, como o Google Maps, para que as coordenadas das passadeiras possam ser representadas num mapa. Para complementar a informação apresentada nos mapas foi utilizada a API OpenCage Geocoding, que permite a adição de informação sobre a país, distrito, rua, entre outras informações pertinentes.

O Gestor de Proximidade, Proximity Manager, deverá gerir a proximidade de peões e viaturas da passadeira, estabelecendo um canal de comunicação bidirecional com o cliente, de forma assíncrona, para que seja possível a emissão de notificações. As posições relativas as estes clientes, Pedestrians e Cars, serão armazenados numa base de dados de memória em Cache, designada por Redis.

Neste sistema, serão utilizados simuladores de carros e peões, Cars e Pedestrians, respetivamente, para que seja possível simular o sistema num contexto real.

### 3.1.1 API Gateway

Analisando a figura 2 é possível ver que o estilo de arquitetura optado foi a orquestração. A API Gateway possui uma interface RESTful, à qual responde a pedidos GET, POST e DELETE, do cliente, recorrendo à biblioteca *Axios*. A API Gatway conhece todos os microserviços da solução, servindo como um intermediário, conhecido também por maestro, ao qual tem a função de reecaminhar o pedido para os respectivos micro-serviços, AuthService, Crosswalks e ProximityManager. Estes micro-serviços serão abordados nos sub-capítulos que se seguem.

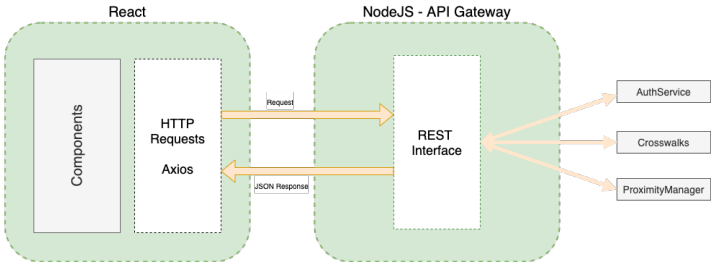


Figura 2: Orquestração de Microserviços

### 3.1.2 Auth Service

Uma das funcionalidades presentes na aplicação é a gestão de passadeiras. Um determinado utilizador, administrador do sistema, pode adicionar ou mesmo remover passadeiras no mapa. Para este efeito, é necessário efetuar autenticação de forma a desbloquear rotas de POST e DELETE na API Gateway e, por consequência, do serviço Crosswalks.

Este micro-serviço, recebe um pedido de autenticação do cliente, passando primeiro pela API Gateway. É importante que o procedimento de autenticação siga esta ordem, uma vez que o serviço se encontra protegido por uma camada Middleware que deverá barrar qualquer pedido proveniente de qualquer Cliente ou Servidor, exceptuando dos micro-serviços aqui implementados. Neste caso, este micro-serviço, deverá aceitar apenas eventuais pedidos dos serviços Crosswalks e do maestro API Gateway. A resposta do serviço deverá ser em JSON, juntamente com o Token, se a validação dos dados de autenticação foi efectuada com sucesso.

A figura 3 representa a arquitectura *core* da aplicação NodeJS, que visa auxiliar a autenticação de Utilizadores.

O serviço foi construído recorrendo ao Express, juntamente com o Mongoose - uma solução que permite implementar um modelo de dados baseado em esquemas. A conjugação destas tecnologias permitem obter um serviço que segue uma arquitectura MVC.

Todos os dados imprescindíveis para a autenticação dos utilizadores, são armazenados num sistema de base de dados NoSQL, designado por MongoDB. As Passwords dos utilizadores são armazenadas em forma de *hash*, recorrendo a um *package* designado por BcryptJS.

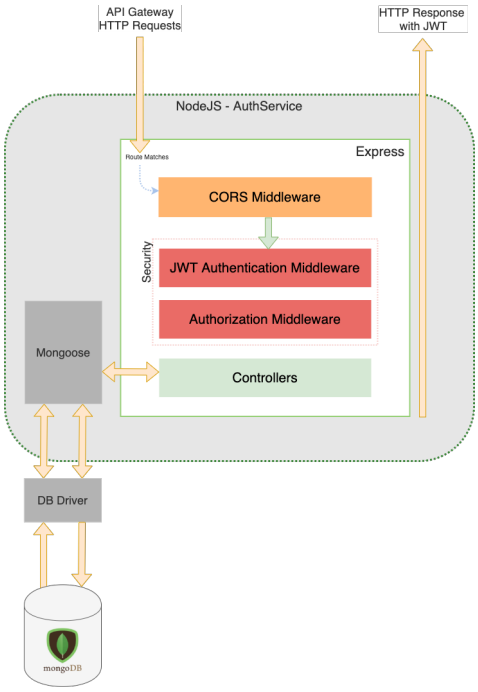


Figura 3: Arquitetura AuthService

### 3.1.3 Crosswalks

Bastante semelhante ao AuthService, o micro-serviço das Crosswalks segue uma Arquitetura MVC, recorrendo ao Express, contudo funciona com uma base de dados diferente, do tipo estruturada, o MySQL. De forma a implementar a camada de modelos neste serviço, foi utilizado um sistema ORM designado por Sequelize.

Outra diferença vísivel neste serviço, é a eliminação de uma camada de segurança, para gerar Json Web Tokens. O serviço API Gateway deve enviar o cookie de autenticação no seu Header para este serviço, que será verificado na camada *Authorization Middleware*, recorrendo ao serviço AuthService. Desta forma, rotas do tipo POST e DELETE deverão ser desbloqueadas.

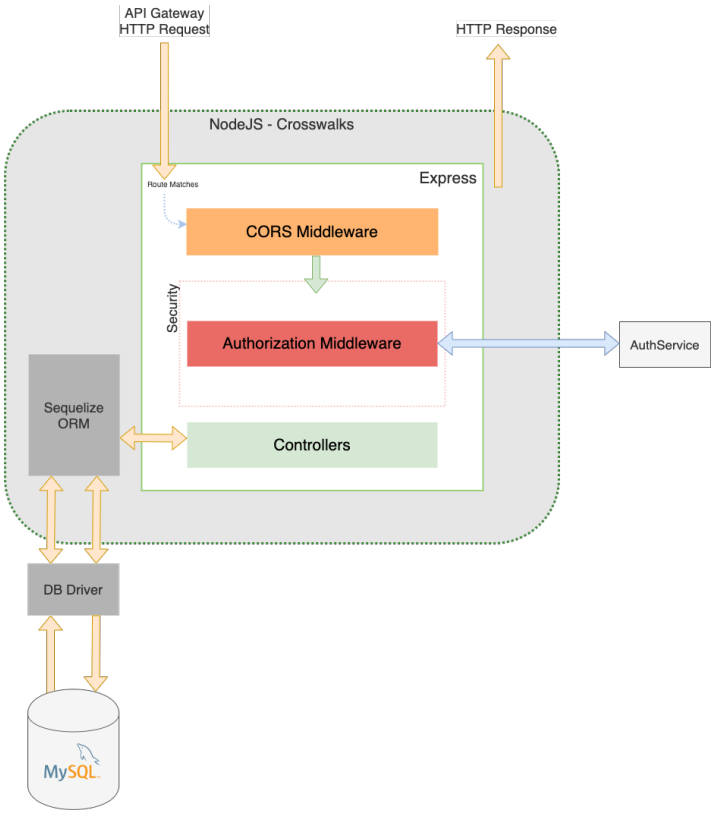


Figura 4: Arquitetura Crosswalks

### 3.1.4 - Proximity Manager

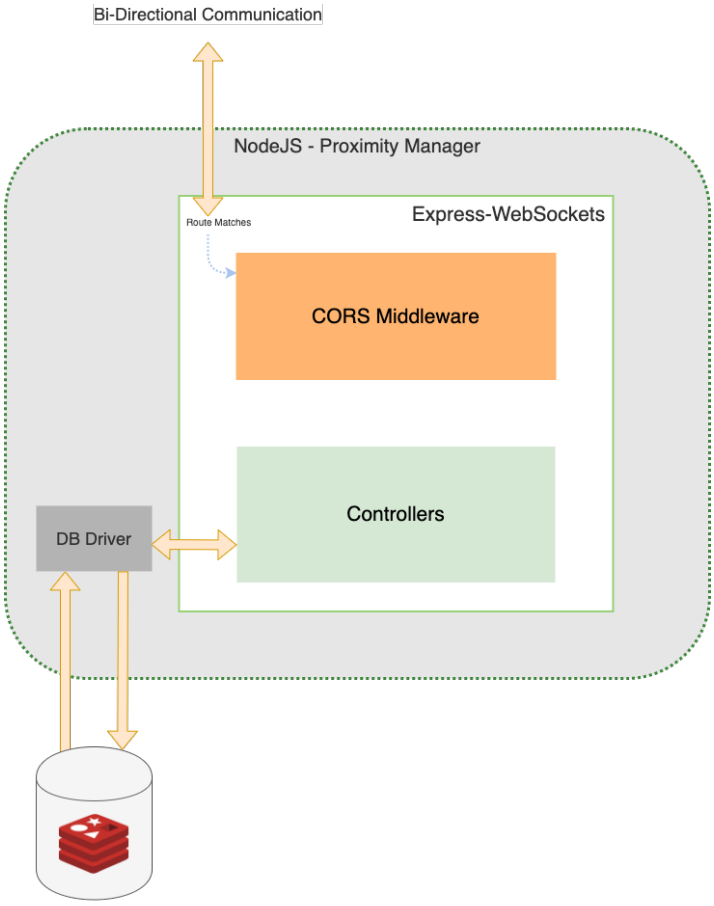


Figura 5: Arquitetura Proximity Manager

O *Proximity Manager* é o responsável pela implementação do use cases. Este micro serviço é responsável por receber as coordenadas dos clientes (pedestres e carros) com o id deste, e devolver um conjunto de notificações, que podem ser direcionadas a cliente que enviou as coordenadas ou a outros clientes que são influenciados pela atualização das coordenadas.

Definimos 3 tipos de notificações, **"red"**, **"orange"** e **"green"**, semelhante ao sistema de semáforos.

Um carro recebe uma notificação **"red"** caso existem pedestres na passadeira e este está muito próximo da passadeira, recebe uma notificação **"orange"** quando existem pedestres na passadeira e o carro ainda não está perto da passadeira, recebe uma notificação **"green"** quando não existem pedestres numa passadeira próxima ou não existe nenhuma passadeira inteligente nas proximidades. Sempre que um carro vá receber uma notificação **"red"** ou **"orange"** os pedestres associados à passadeira inteligente serão notificados como se irá explicar de seguida.

Um pedestre pode receber notificações **"orange"** ou **"green"**, recebe uma notificação **"orange"** quando existe carros muito próximos da passadeira inteligente em que estão a

circular e **"green"** caso contrário. Os pedestres nunca recebem uma notificação **"red"** pois estes têm sempre prioridade de passagem e quem é obrigado a parar é o veículo. Quando um pedestre se aproxima de uma passadeira inteligente informa os restantes veículos próximos.

Todas as notificações são enviadas por sockets ao micro-serviço do backend.

### 3.2 - Deployment

Como é possível verificar pelas imagens arquitecturais anteriores, o sistema foi desenvolvido pensado para um *deployment* orquestrado de micro-serviços.

Para ser efectuado o *deployment* com sucesso da aplicação, é necessário respeitar uma ordem pela qual estes devem iniciar. Inicialmente, devem ser iniciadas as base de dados MongoDB, Redis e MySQL. De seguida, iniciam-se os micro-serviços pela ordem seguinte:

1. AuthService: micro-serviço de autenticação de utilizadores
2. Crosswalks: micro-serviço API de Crosswalks
3. Proximity Manager: micro-serviço de gestor de proximidade das passadeiras
4. API Gateway ou Backend: maestro de todos os micro-serviços. Ponto de ligação entre Cliente e Micro-serviços;
5. Frontend: aplicação em REACT

Entendendo a complexidade no deployment destes serviços um-a-um, criou-se vários Containers das aplicações recorrendo ao Docker. Desta forma, na pasta que agrega todos os micro-serviços, basta correr o seguinte comando para efetuar um deployment rápido de todo o sistema:

```
$ docker-compose down -v
$ docker-compose up --build
```

Contudo, o deployment dos serviços não foi realizado unicamente recorrendo ao Docker. Foi preciso, acrescentar um script em cada serviço, de forma a que estes executassem após os serviços de que dependem tivessem operacionais. O Docker, apesar de iniciar o processo de build dos serviços recorrendo à opção YAML *depends on*, não possui qualquer tipo de controlo sob a ordem pela qual estes são iniciados. Isto torna-se uma adversidade no deployment do sistema, mas foi fácilmente resolvido recorrendo ao script Docker-Compose-Wait [1].

[1] `https://github.com/ufoscout/docker-compose-wait/`

### 3.3. Diagramas de Sequência

Os Diagramas de Sequência permitem identificar as diferentes funcionalidades e forma como o Sistema deverá interagir com os demais Serviços implementados, desta forma serão apenas apresentados os diagramas de sequência mais relevantes.

#### 3.3.1. Utilizador

Tal como já foi mencionado no subcapítulo anterior, o Utilizador poderá aceder aos dados armazenados na base de dados executando pedidos do tipo GET para pesquisa de passadeiras num determinado local.

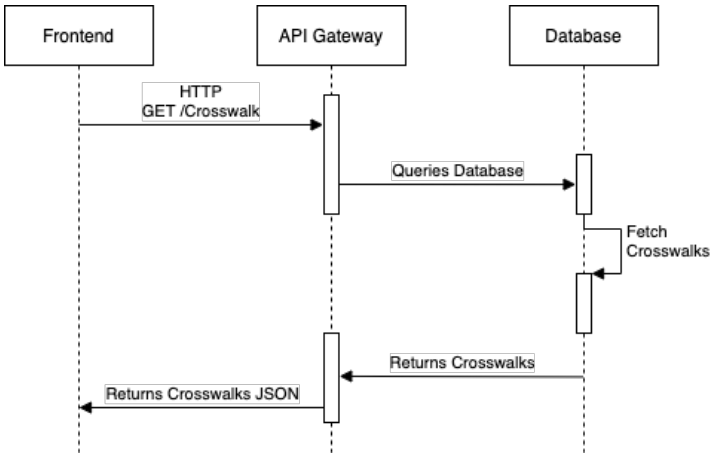


Figura 6: Consulta das Passadeiras

O serviço Frontend, recebendo a resposta da API de Dados com sucesso, deverá mostrar no Mapa a localização da passadeira. É importante referir que o utilizador normal apenas terá acesso ao mapa com todas as passadeiras e ainda poderá visualizar tudo o que se passa em redor das mesmas.

#### 3.3.2. Administrador

O Administrador poderá executar diferentes pedidos na API, uma vez que tenha autenticado com sucesso. Para isto, serão efetuadas várias operações síncronas:

1. O Utilizador introduz as credênciais de acesso no Formulário de autenticação.
2. As credenciais de acesso são enviadas para o serviço de API.
3. A API faz um pedido à base de dados que deverá retornar os dados desse utilizador.
4. São comparadas as credenciais de acesso com as credenciais armazenadas na Base de Dados.
5. Se tudo estiver em conformidade, é gerado um JSON Web Token que deverá ser enviado ao Cliente, e este é redirecionado para a página do mapa.
6. No caso de algum erro, será enviada uma mensagem de Erro em JSON juntamente com um erro de HTTP 403.

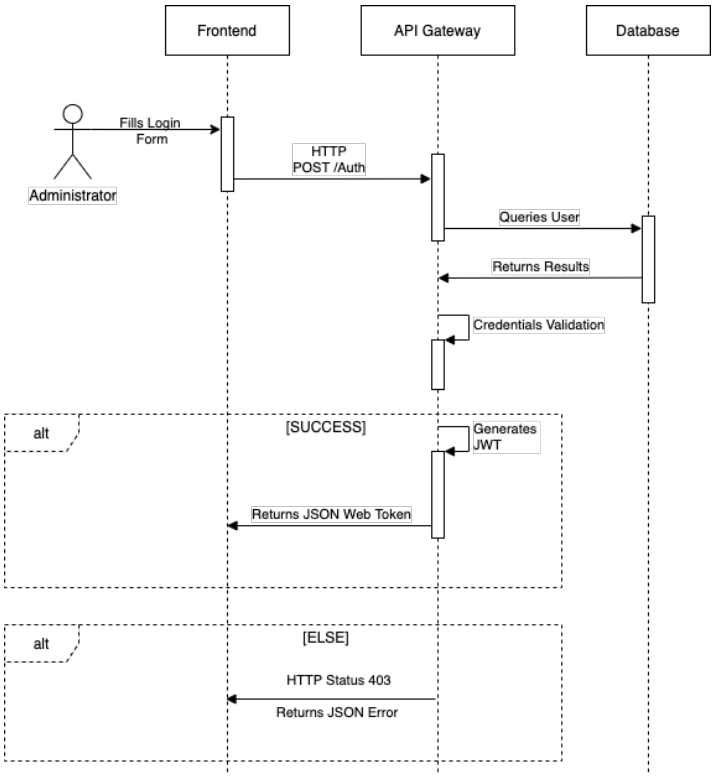


Figura 7: Autenticação do Administrador

Assim que autenticado, o administrador poderá usufruir dos restantes pedidos que a API Rest disponibilizará, entre as quais a criação de novas passadeiras (POST) e remoção de passadeiras (DELETE), sendo que para cada um destes pedidos deverá ser enviado o JWT retornado durante a autenticação.

Este JSON Web Token será sempre validado, antes de ser executado o pedido pela API Gateway.

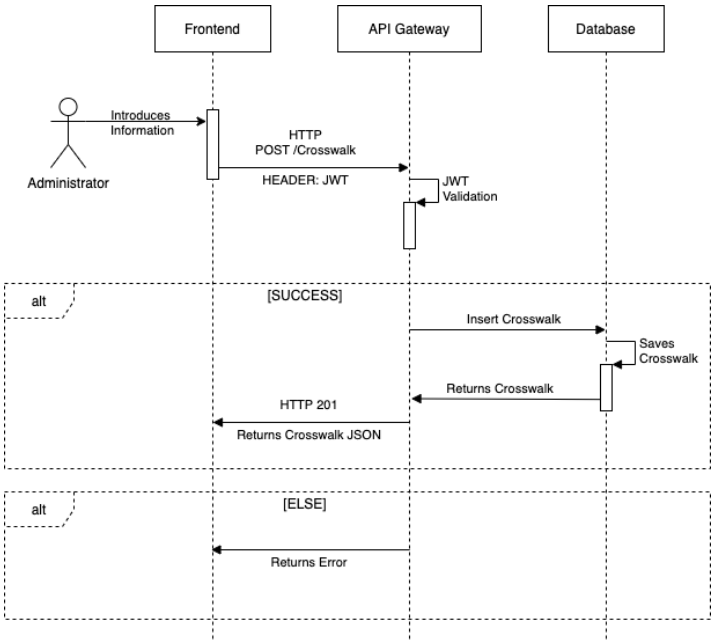


Figura 8: Criação da Crosswalk

### 3.3.3. Gestor de Proximidade

O Gestor de Proximidade é um serviço que deverá calcular a distância de objetos na proximidade das passadeiras.

O Utilizador pode ser instanciado como Carro ou Pedestre. A aplicação conhece previamente as passadeiras na proximidade, recorrendo à API de Dados e ao serviço externo de Mapas. A aplicação cliente calcula a proximidade do seu utilizador à passadeira, devendo começar a emitir as coordenadas quando se encontra num raio de 150 metros da passadeira, permitindo desta forma ao gestor de proximidade emitir notificações para os restantes clientes na proximidade que tenham estabelecido um canal bidirecional com este serviço, estas notificações são responsáveis por indicar se é seguro ou não atravessar a passadeira.

Trata-se, portanto, de um serviço com funções de comunicação de forma assíncrona.

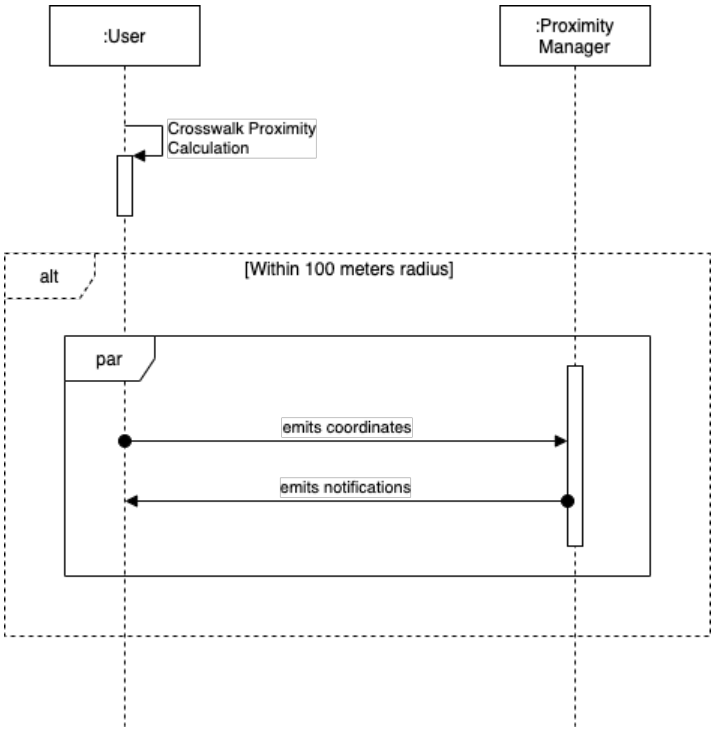


Figura 9: Gestor de Proximidade

De maneira a explicar a forma como as passadeiras e os carros são geridos neste microserviço foi desenvolvido diagrama de sequência da figura 10.

Analisando a figura em questão pode verificar-se a existência de 3 casos principais, isto é, comportamento com passadeiras que estão num raio de 100m, comportamento com passadeiras que estão num raio de 40m e passadeiras que estão depois dos 100m.

No primeiro ciclo são tratadas as passadeiras que estão a menos de 100m do carro verificando que caso não existam pedestres e o seu estado não seja "green", o seu estado é alterado para "green", por outro lado caso existam pedestres e o seu estado ainda não seja "orange"o seu estado é alterado para "orange". No segundo ciclo é tratado o comportamento com as passadeiras que estão mais próximas do carro, caso não existam pedestres lá e se o seu estado não for "green"o estado é alterado para "green", em contrapartida se existirem pedestres lá e se o estado do carro ainda não for "red", este passa a "red". As passadeiras que estão depois dos 100m não são tidas em conta.

De notar que todas estas alterações são enviadas em forma de notificação para os simuladores e para o frontend do sistema.

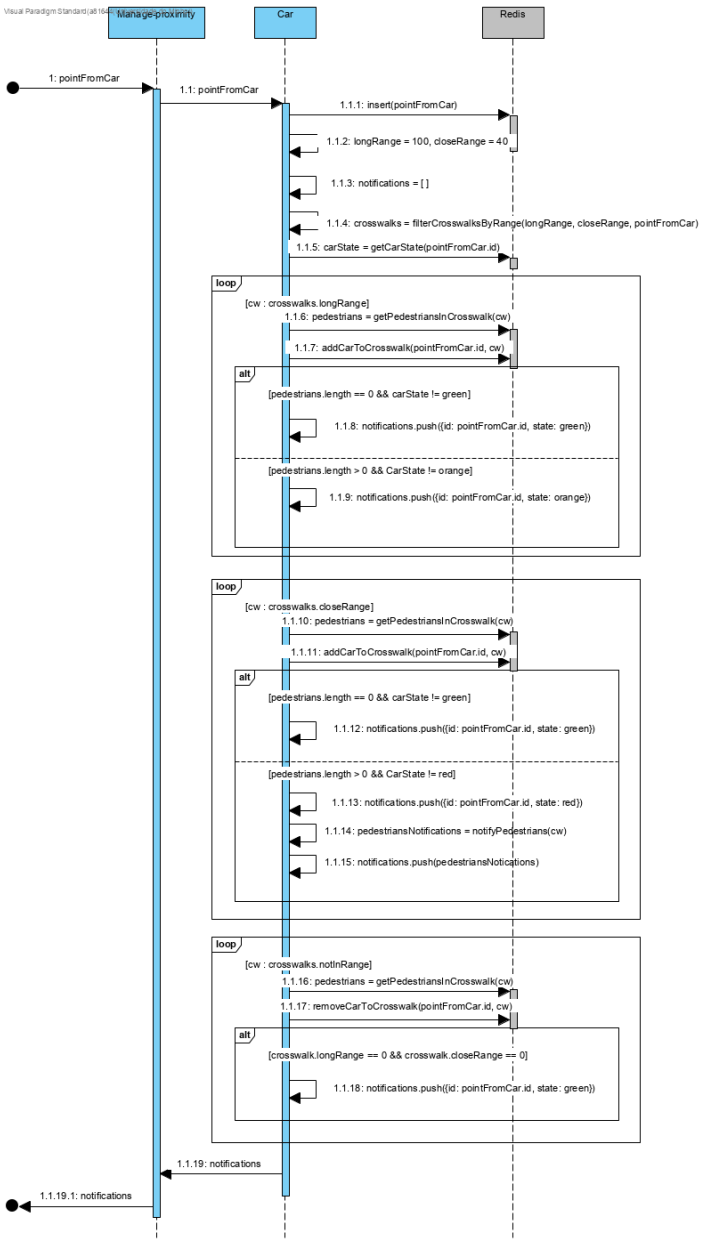


Figura 10: Gestor de Proximidade - carros

De forma análoga é feito o mesmo diagrama para o caso dos pedestres, caso estes se encontrem na passadeira têm sempre prioridade sobre os carros. Neste são alterados os raios, uma vez que, a velocidade de um pedestre é menor do que a de um carro.

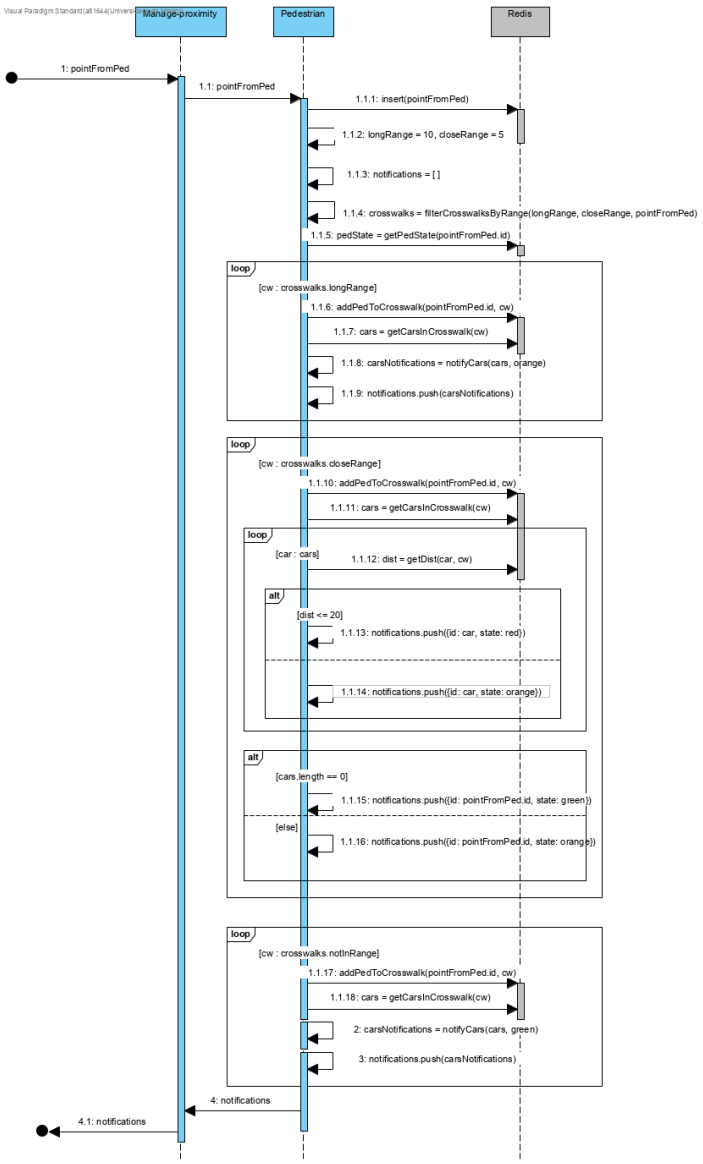


Figura 11: Gestor de Proximidade - pedestres

# 4. Interface

Depois de explicado todo o funcionamento geral, serão apresentadas em seguida algumas imagens indicativas de como utilizar o sistema e, como referido anteriormente, a interface da aplicação foi construída com recurso ao React.

Em toda a aplicação está presente uma barra de navegação que permite aos seus utilizadores aceder a qualquer uma das páginas em qualquer ponto do sistema.

## 4.1 Página Principal - Mapa

A página principal apresentará todas as passadeiras que estão presentes no sistema através do icon apresentado na figura 12a, se existirem carros ou pedestres a utilizar passadeiras estes serão apresentados com recurso aos icons 12b e 12c, respetivamente.

(a) Icon passadeira

(b) Icon carro

(c) Icon pedestre

Figura 12: Icons

Desta forma o utilizador consegue acompanhar tudo o que se está a passar em qualquer ponto do mapa e identificando cada um dos intervienientes.

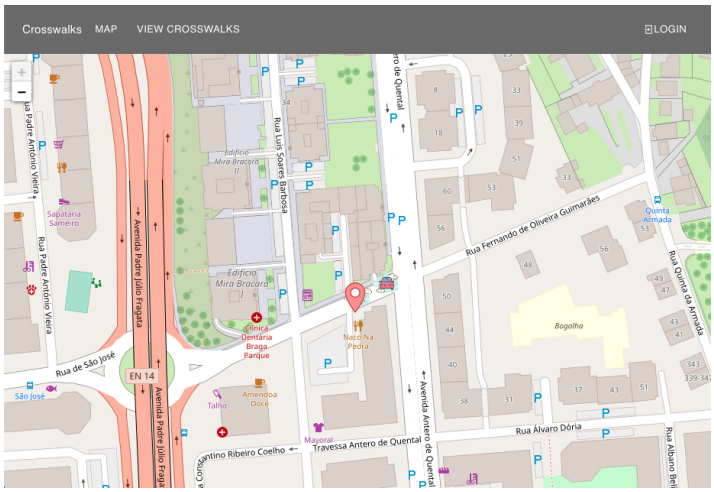


Figura 13: Visualização de Componentes no Mapa

## 4.2 Administradores

### 4.2.1 Login

Só os administradores do sistema com credenciais previamente colocadas na base de dados MongoDB, é que poderão realizar a ação de login para aceder a funcionalidades mais avançadas.

Para efetuar o login basta introduzir o seu nome de utilizador e a sua password. De forma a ser possível testar a plataforma, existe uma conta *default* já criada, cujo o *username* e *password* são ***admin***.

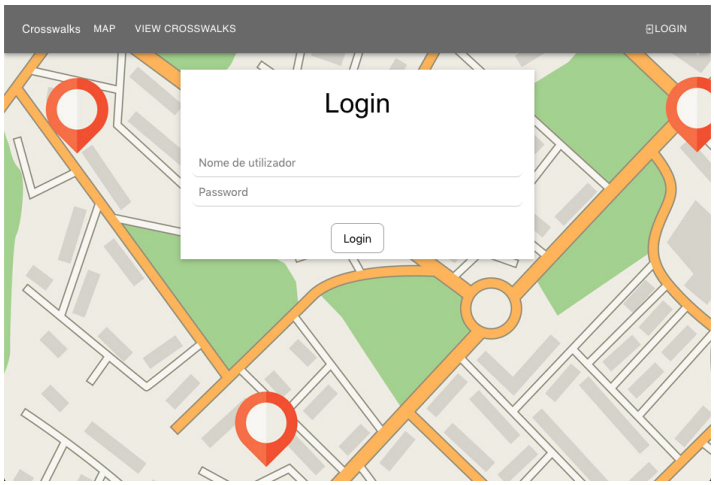


Figura 14: Página de Login

### 4.2.2 Adicionar passadeira

Assim como anteriormente referido, só os administradores podem adicionar novas passadeiras ao sistema para garantir que este se mantém coerente e com todas as informações corretas para que os utilizadores possam usufruir do sistema com segurança.

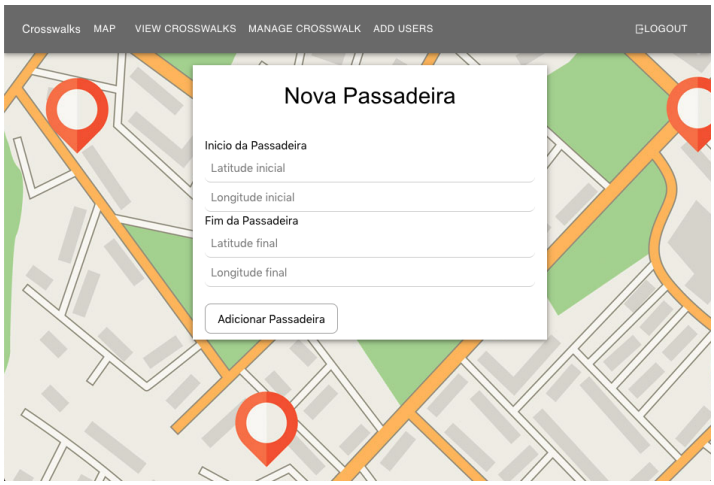


Figura 15: Nova Passadeira

### 4.2.3 Adicionar Utilizadores

Os administradores possuem ainda o possibilidade de registar novos utilizadores para desempenhar papeis de administradores

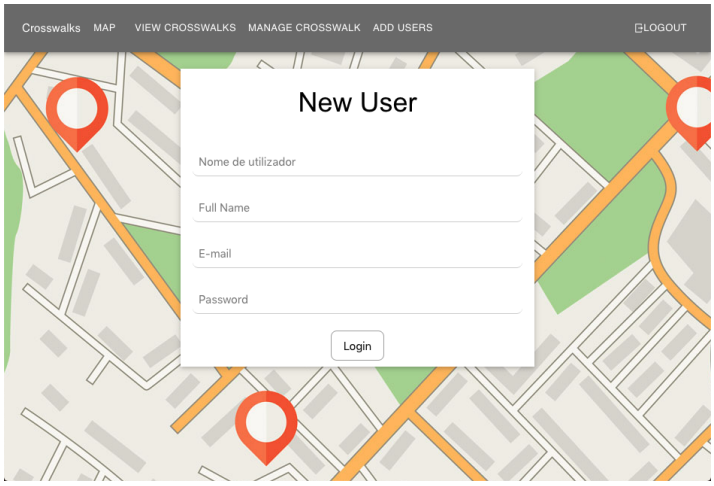


Figura 16: Novo Utilizador

### 4.2.4 Eliminar Passadeiras

Existe ainda a opção dos administradores eliminarem as passadeiras do sistema.

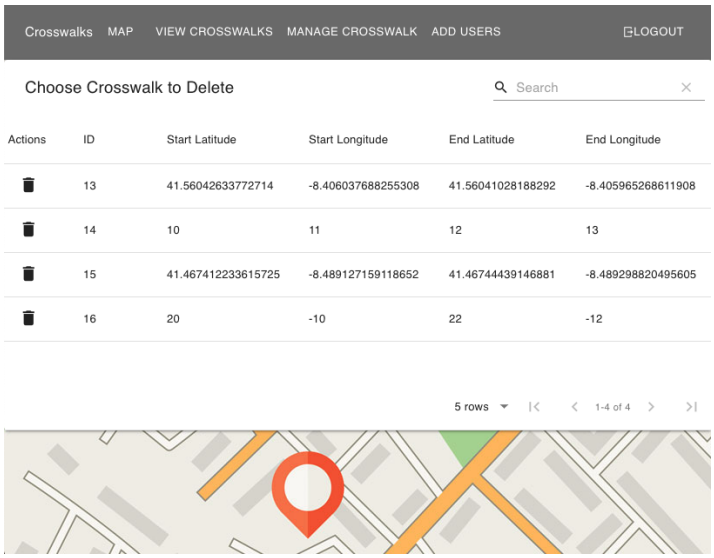


Figura 17: Eliminação de Passadeiras

## 4.3 Ver passadeiras

Qualquer utilizador do sistema pode ver todas as passadeiras inseridas no sistema e onde estas se localizam.

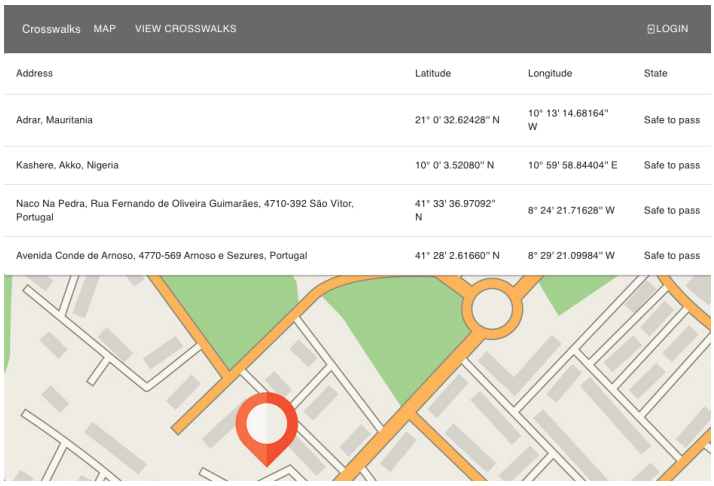


Figura 18: Visualização de Passadeiras Existentes

# 5. Simuladores

Para que fosse possível reproduzir a movimentação de pedestres e carros foi necessária a criação de simuladores. Estes simuladores apenas são responsáveis por emitir as coordenadas atuais de cada interveniente.

Esta comunicação é realizada através de um canal bidirecional, que é estabelecido a partir do momento em que o utilizador acede à página onde se encontra o mapa. Este canal bidirecional é implementado através de Web Sockets, este é mais um pacote fornecido pelo NPM.

Chegado a esta página e depois da conexão estabelecida é possível fazer o cruzamento de informação entre pedestres e carros. Aqui podem surgir vários cenários:

1. O primeiro cenário elaborado trata-se do mais simples onde o carro começa longe da crosswalk e o pedestre perto e este atravessa facilmente a passadeira e o carro também não necessita de parar.

2. O segundo cenário é inverso do primeira quem começa mais longe é o pedestre e o carro mais perto, passando os dois por esta sem qualquer problema

3. O último cenário tem em conta a situação mais crítica, ou seja, quando ambos estão próximos da passadeira, o que acontece aqui é que o carro espera o pedestre passar a passadeira e depois arranca com velocidade reduzida, voltando depois à sua velocidade normal quando se encontrar mais longe deste.

# 6. Conclusão

No início do trabalho prático começou-se por identificar todos os micro-serviços, como estes se deveriam dispor na arquitectura e como estes interagem entre si. Desta forma, surge uma arquitectura final, onde se pode ver que foi contornado um dos principais problemas das arquitecturas de micro-serviços, a dependência de uma única base de dados, o que não acontece aqui dada a existência de uma base de dados para cada serviço.

Em seguida passou-se ao desenvolvimento dos vários serviços e componentes da arquitectura onde foram sentidas as principais dificuldades. Estas surgem, sobretudo, na utilização de novas ferramentas, *React* e *Redis*, ambas porque não existia qualquer tipo de experiência prévia e, por isso, houve a necessidade de perceber o funcionamento geral das mesmas. Quanto à interface do sistema, o objectivo principal era que esta fosse simples e de fácil aprendizagem para que o utilizador final não tenha de perder muito tempo a familiarizar-se com o sistema. Surgiram também algumas dificuldades no que toca ao *deployment* através do *Docker*, isto é, a existência de vários micro-serviços tornou mais complicada esta actividade uma vez que existem várias dependências entre eles e era necessário garantir que estes conseguiam comunicar entre si, e só iniciam após os serviços de que dependem estarem ativos e operacionais. Esta última adversidade, tal como foi abordado no capítulo 3.2, foi resolvida utilizando um *script shell* que permite bloquear a execução de um determinado serviço.

Com a elaboração do presente sistema é percetível a dificuldade de construir um sistema crítico de forma robusta e sem falhas. São, por isso, muito importantes ferramentas como WebRatio, VisualParadigm e outras, que permitem fazer uma análise geral do sistema para que sejam encontradas as imprecisões.

A construção de software sem erros é imperativa principalmente quando estão em jogo vidas humanas.