INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

JiTT.travel Web Platform

Relatório de estágio para a obtenção do grau de Mestre em Informática e Sistemas

Autor

Rui Pedro Barreto Amado Ferreira

Orientação

Professor Doutor Francisco Pereira

Engenheiro Miguel Monteiro

Coimbra, Maio de 2019

“A melhor maneira de prever o futuro é inventá-lo.”

Alan Kay, cientista de computação, 1971

Abstract

The Internet has completely changed the way we look at life and how we move around the world. Travelling (inside and outside of the country) has become much more frequent, but its time planning has become of major importance. The way we discover new places (sceneries, monuments, cities…) has completely changed: the trips organized by agencies have become obsolete with the appearance of personalized trips, with family or friends, in which we want to plan our time the way we believe is the best. The web applications presented in this work aims to create routes which avoid crowded places thus make these places' discovery as fast and orderly as possible, by avoiding ticket line waiting and overcrowding. Individually or in small groups, we can plan our time in a better way so we can explore more and better.

Keywords: Web, Application, Tourism, Classification, Itineraries, Crowd, Image Recognition

Resumo

A *Internet* alterou completamente a forma como encaramos a vida e como nos movimentamos no mundo. As viagens (dentro e fora do país) são cada vez mais frequentes, mas o tempo parece ser cada vez importante. A forma como nos propomos conhecer novos lugares (espaços, monumentos, cidades…) tornou-se fulcral: as viagens organizadas por agências deram lugar a viagens individualizadas, em família ou com amigos, nas quais queremos gerir o tempo da forma que nos parece mais adequada. A plataforma digital aqui apresentada visa tornar estas visitas a locais importantes mais célere e mais profícua, apresentando rotas para visitas e diminuindo o tempo de espera para cada uma delas: individualmente ou em pequenos grupos, podemos organizar melhor o nosso tempo e conhecer mais… e melhor…

Palavras Chave: Web, Aplicação, Turismo, Classificação, Itinerários, Multidão, Reconhecimento de Imagem.

Agradecimentos

Não teria conseguido realizar o estágio nem tão pouco este relatório com êxito sem a preciosa colaboração de algumas pessoas, que aqui passo a referir:

O Professor Francisco Pereira, coordenador do estágio e, consequentemente, deste relatório: um muito obrigado pela sua disponibilidade, pelos seus conselhos e pelo seu acompanhamento sistemático, tantas vezes, por questões de horários, com prejuízo da sua vida pessoal.

O Miguel Monteiro, Coordenador do estágio na empresa iClio: um grande bem-haja pela aceitação deste projeto, pelo desafio lançado e pela atenção que sempre m e concedeu, envidando todos os esforços no sentido de garantir as necessárias condições de trabalho e o desenvolvimento do projeto.

Todos os colegas da empresa iClio, por me terem recebido de braços abertos e me terem integrado, com toda a facilidade, no seu dia a dia.

Também os meus colegas de curso, que, sempre que necessário, facultaram todo o apoio, moral e físico, que foi necessário.

O Doutor Humberto Rendeiro – Ruinas de Conimbriga, que me ajudou a encontrar uma forma de aceder a imagens que tanta falta me faziam para o desenvolvimento da aplicação.

Todos os processos que, ao longo do meu percurso escolar, procuraram "ensinar-me pescar" em vez de me "darem o peixe", já que me facultaram as ferramentas necessárias ao meu desenvolvimento pessoal, bem como a capacidade de procurar soluções para os problemas que, no dia a dia, vão surgindo.

Os meus amigos pessoais, que passavam o tempo a ouvir os meus "não posso sair hoje, porque tenho que trabalhar" … e que se mantiveram do meu lado!

Finalmente, mas não menos importante, os meus pais, que "aguentaram" todo o stresse, ouviram todas as dúvidas, me aconselharam quanto aos problemas que me iam surgindo, bem como à forma de os ultrapassar, e me deram todo o apoio de que necessitava para prosseguir este trabalho.

A todos, o meu sincero "muito obrigado"!

Índice

Lista de Figuras

Lista de Tabelas

1. Introdução

1.1. Enquadramento

Num mundo em que as viagens de longa distância são cada vez mais baratas, têm menor duração e maior conforto, tornou-se possível, para cada vez mais pessoas, explorar sítios distantes em viagens inesquecíveis, de forma personalizada e sem os constrangimentos inerentes a viagens organizadas, por exemplo, através de agências de viagens. Entre estes constrangimentos estão, seguramente, a questão dos horários a cumprir, bem como, muitas vezes, a necessidade de fazer visitas em grupo.

Para melhorar estas viagens, nomeadamente através da utilização dos nossos *smartphones* como objetos pessoais e indispensáveis nas nossas vidas, surgiram diversas aplicações no mercado, que servem de ajuda e de guia importante para esses mesmos itinerários pessoais de viagens. Estas aplicações pretendem, fundamentalmente, tornar as viagens em geral e as visitas mais autónomas, mais independentes e, consequentemente, mais confortáveis a determinadas cidades ou lugares específicos para aqueles que assim o desejam.

No entanto, estas aplicações funcionam, por norma, com recurso a itinerários pré-definidos, o que as leva a aconselhar todos os utilizadores a visitar uma lista de pontos de interesse pela mesma ordem, piorando a experiência dos seus utilizadores, já que levam à sobreocupação dos diferentes (mas, insistimos, predeterminados e sempre comuns) pontos de interesse e ao aumento das filas para entradas nos ou visitas aos mesmos.

1.2. Objetivos

Como solução para este problema, ao longo do estágio curricular aqui abordado, foi desenvolvida uma aplicação *web* que terá como objetivo criar itinerários pessoais para cada utilizador, tendo em consideração aspetos importantes, como a taxa de ocupação de um ponto de interesse através de reconhecimento de vídeo em tempo real ou os pontos de interesse escolhidos pelo utilizador, otimizando sempre a rota para o menor caminho a percorrer.

Para tal, foi proposta a criação de uma ferramenta de criação de rotas turísticas para o utilizador. Tendo em conta que este é um mercado com um número muito alto de soluções e com

um nível alto de concorrência, e na tentativa de que a ferramenta desenvolvida se destacasse como inovadora, foi considerado adicionar o controlo de densidade populacional à mesma.

Ao adicionar o controlo de densidade populacional à ferramenta desenvolvida, espera-se que o utilizador consiga criar rotas para turismo que evite pontos de interesse sobrelotados, diminuindo, assim, o tempo de espera em filas, nomeadamente para compra de bilhetes de entrada, e proporcionando uma melhor experiência ao utilizador em cada ponto de interesse.

A ferramenta desenvolvida tem como objetivo funcionar como uma *RESTful API*, que permite a comunicação entre os clientes e o servidor, de forma a restringir ao máximo o acesso ao servidor por parte do utilizador. Trata-se, perante o atrás exposto, de uma inovação que pretende, de forma clara e inequívoca, facilitar a vida do viajante, levando-o a visitar, à sua inteira consideração, o(s) lugar(es) ou monumento(s) que quer visitar, seguindo o seu próprio itinerário em vez de itinerários predefinidos, ao seu próprio ritmo e de acordo com a sua própria disponibilidade ou vontade.

1.3. Enquadramento Institucional

De facto, quando, no início do ano curricular do Mestrado em Informática e Sistemas do Instituto Superior de Engenharia de Coimbra, no ramo de especialização de Tecnologias da Informação e do Conhecimento em 2017/2018, nos colocámos perante a possibilidade de fazer um estágio na empresa iClio, a motivação foi enorme. Trata-se de uma empresa que foi criada por quatro jovens empreendedores (Alexandre Pinto, Joaquim Carvalho, Margarida Gomes e Ana Isabel Ribeiro), para quem a universidade de Coimbra, onde trabalhavam como professores ou como investigadores, já não parecia chegar: queriam ir mais além e decidiram criar um projeto próprio e pensaram num produto para contar histórias. Foi com este pressuposto que lançaram a iClio, cujo primeiro produto foi a aplicação para telemóvel Just in Time Tourist (JiTT), uma aplicação que desempenha o papel de guia turístico à imagem e à medida do seu utilizador, que pode passear por cidades de diferentes países (Barcelona, Nova Iorque, Paris…) aproveitando o tempo com história, histórias e dicas relativamente aos locais que visita: desta forma, cada turista pode sentir-se único e abstrair-se dos "pacotes" de viagens oferecidos pelas agências. Com o investimento da Portugal Ventures, a iClio cresceu: apostou no marketing, apostou em produtos inovadores e apostou em ganhar um novo mercado: o dos turistas sem tempo a perder e com histórias para ouvir, os turistas que querem não só ver, como ver melhor e conhecer a história e as histórias dos lugares que visita.

Esta empresa nasceu em Portugal, mais concretamente em Coimbra, onde tem a sua sede, mas já tem filiais nos EUA, no Canadá e em muitos países anglófonos, pretendendo, também, chegar ao mercado chinês, um dos mais importantes no que ao turismo diz respeito.

1.4. Estrutura do relatório

O trabalho que aqui apresentamos organiza-se da seguinte forma:

2 - Arquitetura da aplicação desenvolvida: nesta parte, faz-se a explicação da criação/desenvolvimento da aplicação que serviu de base a este relatório. Esta explicação é apresentada através de um esquema, no qual podemos encontrar os quatro módulos principais da mesma.

3 - Revisão da literatura existente.: referem-se plataformas digitais já existentes que respondem a algumas das necessidades do mercado de viagens/visitas a determinadas, cidades ou monumentos, bem como plataformas que permitem, através da captação de imagens, detetar situações e/ou solucionar problemas.

4 - Referência às diferentes tecnologias utilizadas para a criação da aplicação por nós criada e aqui descrita.

5 - Menção dos modelos usados como base de trabalho e sua breve descrição.

6 - Descrição de todos os passos que levaram à criação desta aplicação, no que ao servidor diz respeito.

7 - Descrição dos modelos de algoritmos de rotas, do trabalho de reconhecimento de imagens e de vídeos, da criação do *dataset*, do treino e funcionamento do modelo, dos algoritmos de rotas (personalizados e automático), da criação da *frontend* e das diferentes componentes da aplicação.

8 - Neste ponto, referem-se as possibilidades de implementação futura da aplicação criada e desenvolvida, bem como as suas vantagens em termos de turismo. Chama-se a atenção para as potencialidades que apresenta, em termos de turismo nacional e estrangeiro.

9 -Trata-se da conclusão final, na qual se referem alguns dos problemas encontrados e superados, bem como as potencialidades que vemos no uso desta aplicação.

2. Arquitetura da aplicação

A aplicação desenvolvida é ser constituída por quatro módulos principais, que serão descritos ao longo deste relatório. Estes quatro módulos são a *Frontend*, que será a página em que o utilizador irá interagir com as rotas criadas dentro do servidor, a *Backend*, que será a componente que contém a base de dados e o servidor que está à espera de pedidos da *Frontend*, o modelo de previsão de taxa de ocupação de um ponto de interesse, que será uma componente individual que irá comunicar diretamente com o servidor, dando, assim, um funcionamento autónomo à aplicação, e a criação de itinerários, que será a aplicação dos algoritmos de rotas na *Frontend* (com o intuito de diminuir a quantidade de processamento necessária para o servidor, tornando o mesmo mais livre para aceitar pedidos de outros utilizadores), que embora seja uma parte da *Frontend*, será descrito como uma componente individual devido à sua complexidade.

As diferentes componentes da aplicação desenvolvida e a relação entre as mesmas pode ser visualizada na seguinte figura:

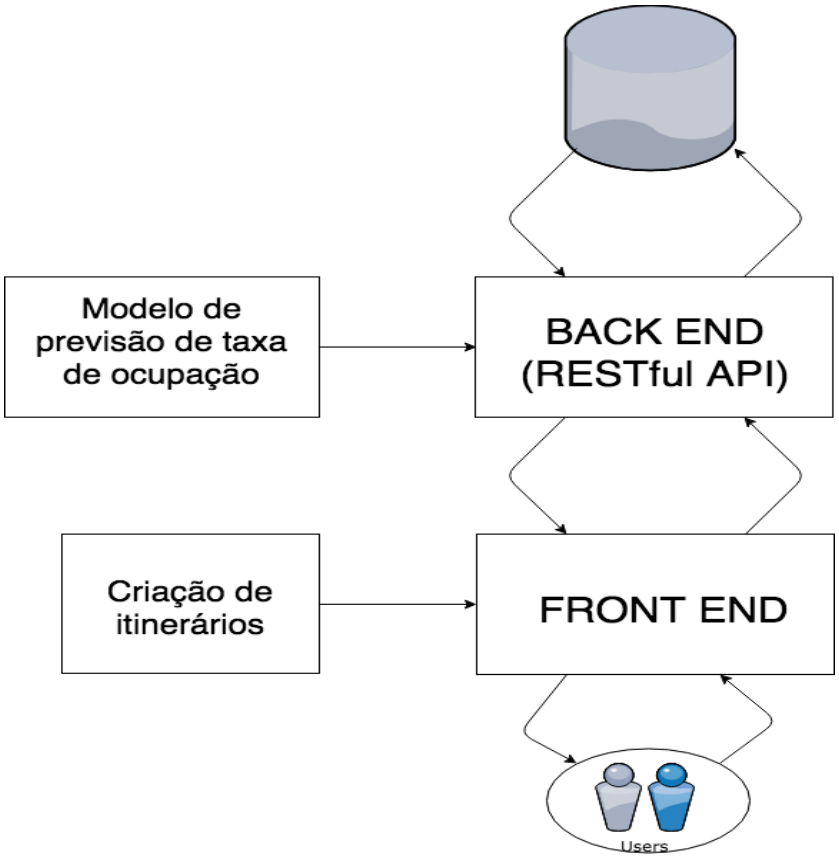


Figura 1 - Funcionamento da aplicação desenvolvida

Na imagem anterior, podem ser distinguidos 4 módulos principais a desenvolver para a criação desta aplicação web, sendo os mesmos:

- A base de dados;
- O servidor;
- A interface disponibilizada ao utilizador, designada neste relatório de Frontend;
- Os modelos de classificação de dados.

A aplicação deverá ser criada tendo uma série de requisitos em mente, sendo os mesmos apresentados de seguida:

- Existência de dois tipos de utilizadores: os administradores, que poderão criar, editar e remover dados da base de dados, e os utilizadores normais, que poderão criar rotas turísticas para visitar as cidades escolhidas;
- A base de dados deverá guardar informação relativa às cidades para as quais é possível criar rotas, aos pontos de interesse presentes nas mesmas, às câmaras de videovigilância e as previsões da densidade populacional de cada um desses pontos de interesse e aos utilizadores da plataforma (para autenticação e sessão);
- Deverá ser possível para um utilizador normal criar rotas com dois tipos de funcionamento distintos, sendo o primeiro uma rota baseada num tempo inserido pelo utilizador e pela posição inicial do mesmo e o segundo uma rota personalizada baseada nas categorias escolhidas pelo utilizador para pontos de interesse a visitar, permitindo ao mesmo criar uma rota do seu agrado pessoal;
- Os modelos de classificação da densidade populacional de um ponto de interesse e os algoritmos de rotas implementados deverão ser independentes e simples de implementar futuramente noutras aplicações.

3. Revisão da literatura

A aplicação desenvolvida, apesar de apresentar características pioneiras, encontra-se dentro da categoria de aplicações de guias turísticos / rotas, na qual se pode encontrar um vasto leque de soluções.

De entre as soluções atualmente disponíveis no mercado, há algumas que, pelas suas caraterísticas, funcionalidades e frequência de utilização, se destacam de todas as outras. Falamos da *Yelp*, da *TripAdvisor* e da *Kayak*, que procuraremos descrever, de forma breve e concisa.

Há, também, soluções que são conhecidas, mas que não estão disponíveis ao público, como por exemplo o sistema de videovigilância da China, onde existem redes de câmaras e métodos de reconhecimento de imagem extremamente complexos, mas não acessíveis ao público. Porém, deve referir-se que, muito embora não seja, por enquanto, uma solução que possa ser adquirida, é, sem dúvida, uma solução pioneira na área e merece ser mencionada neste projeto.

Apesar de não serem soluções disponíveis no mercado, seria também importante mencionar artigos disponíveis *online*, nomeadamente a classificação de afluência de trânsito numa estrada, etc.

3.1. Yelp

A Yelp é uma empresa multinacional fundada em 2004 e sediada na Califórnia, mais concretamente em São Francisco, que apresenta várias aplicações, maioritariamente vocacionados para a avaliação de estabelecimentos comerciais. A par disso, procede, também, ao treino de pequenas empresas, no sentido de as capacitar para melhor satisfazer as necessidades dos seus clientes, nomeadamente usando as avaliações feitas pelos seus usuários para a promoção de melhorias do serviço prestado por essas mesmas empresas.

Trata-se de uma multinacional que, como tantas outras, começou de forma mais restrita, baseando o seu serviço em referências que os seus usuários lhe forneciam por email, de forma voluntária. Porém, esta estratégia não se revelou muito profícua e a Yelp acabou por se metamorfosear, optando por apresentar uma plataforma na qual constam avaliações feitas, de

forma voluntária, pelos clientes de empresas. Foi-se expandindo bastante, até chegar, em 2010, a mais de 4,5 milhões de avaliações no seu *site*.

Pelo meio, em 2009, em função do volume de negócios alcançado, a Google tentou adquiri-la, mas essa intenção nunca foi concretizada. Entre 2009 e 2012, expandiu-se a outros continentes, nomeadamente Europa e Ásia, tendo aumentado exponencialmente o seu alcance empresarial. Tornou-se, em 2012, uma empresa pública e foi cotada em bolsa a partir de 2014, alcançando, nesse mesmo ano e com mais de 57 milhões de avaliações, uma média de 132 milhões de visitantes por mês.

A plataforma da Yelp não é mais do que uma rede social, particularmente dedicada à avaliação de negócios locais. Desta forma, começa por surgir em capitais e grandes metrópoles, onde há mais usuários possíveis. Esta plataforma apresenta páginas distintas, destinadas a cada localidade e com um conteúdo personalizado, que vai de restaurantes a escolas, de empresas a outros serviços. Nelas, o usuário pode colocar comentários e avaliações sobre empresas, atribuindo uma avaliação por estrelas. Quanto às empresas, têm a possibilidade de atualizar as suas informações (contactos, horários, ofertas…).

Porém, a Yelp não se limita a avaliações e comentários, oferecendo também, aos seus utilizadores, meios que lhes permitem planear eventos ou discutir questões do dia a dia.

A maior parte dos seus utilizadores acede a este serviço através de dispositivos móveis, aumentando a capacidade de utilização do mesmo serviço que, entretanto, foi sendo atualizado e melhorado, oferecendo cada vez mais serviços e potencialidades (serviço de check-in, reservas em restaurantes, encomendas de comida para casa…).

Como qualquer empresa do género, a sua rentabilidade provém das publicidades que apresenta no seu *site*. E quanto maior é o número de visitantes, maior é, evidentemente, o número de empresas que aí procuram colocar, mediante pagamento, a sua publicidade.

3.2. TripAdvisor

Como o próprio nome indica, TripAdvisor.com é uma plataforma *online*, diretamente vocacionada para as viagens e para o turismo, facultando informações e ajudando os seus utilizadores aquando do planeamento/realização de uma viagem.

Fundado em 2000, inclui, atualmente, fóruns interativos sobre viagens e assumiu protagonismo pela inovação, já que foi dos primeiros a apresentar conteúdos gerados pelos utilizadores, sendo mesmo estes que fornecem a maior parte dos conteúdos aí publicados.

Assume-se, atualmente, como o principal serviço usado pelos viajantes a nível mundial, tendo mais de 32 milhões de usuários e mais de 100 milhões de comentários e avaliações relativos a hotéis, restaurantes, monumentos, pontos de interesse…

Como referíamos relativamente à Yelp, também aqui a sua rentabilidade advém das publicidades que apresenta no seu *site*, sendo que, mais uma vez, quanto maior é o número de visitantes, maior é, naturalmente, o número de empresas que aí procuram colocar, mediante pagamento, a sua publicidade.

3.3. Kayak

A Kayak é uma empresa fundada em 2004 e sediada nos Estados Unidos que apresenta como principal caraterística a de ser um motor de metapesquisa de viagens. Trata-se de uma plataforma que recebe, anualmente, mais de um bilião de pesquisas de viagens, sendo que as aplicações que oferece para *iOS* e para o sistema *Android* também contabilizam mais de 40 milhões de *downloads* anuais.

Em 2010, a Kayak expandiu-se e adquiriu a Swoodoo, uma das maiores plataformas de viagens da Alemanha, bem como a Checkflix, uma plataforma austríaca destinada ao mesmo serviço. Atualmente, está presente em mais de 30 países, entre os quais se incluem Alemanha, Argentina, Áustria, Brasil, Espanha, Estados Unidos, França, Itália, México, Portugal e Reino Unido.

Apesar de estar "apenas" em mais de 30 mercados, isso não lhe retira o caráter universal, já que, em qualquer país do mundo, podemos aceder, via internet, aos serviços que oferece.

À semelhança das anteriores, também o seu lucro advém das publicidades que recebe na sua página e que, como tal, são apresentadas a todos os utilizadores.

3.4. Sistema de Videovigilância Chinês

A China é, efetivamente, uma das grandes potências mundiais, nomeadamente em termos de desenvolvimento tecnológico, e criou um sistema de videovigilância que visa baixar a criminalidade nas ruas. Trata-se do maior e mais moderno sistema inteligente de videovigilância e permite o reconhecimento facial. Pode ser usado como forma de reconhecimento facial permite

saber, de forma imediata, se a pessoa em causa corresponde, ou não, à que consta no seu documento nacional de identificação.

Segundo a BBC, haverá cerca de 170 milhões de câmaras de vigilância espalhadas por todo o país, prevendo-se que este número atinja os 400 milhões dentro de muito pouco tempo. Tratando-se de um circuito fechado de televisão, este sistema cruza informações diversas, que vão desde o grau de parentesco da pessoa identificada, ao veículo que conduz, passando pelas pessoas com quem manteve contactos, entre outros não especificados.

Para complementar este sistema de reconhecimento facial, a China criou, também, uma base de dados e um *software* capaz de fazer o reconhecimento de voz, o que poderá aumentar, ainda mais, o nível de segurança nas ruas.

Como qualquer outro sistema de vigilância, apresenta vantagens, mas também desvantagens. Entre estas últimas está, sobretudo, a questão da política de privacidade e de proteção de dados. Quanto a este aspeto, as autoridades referem que não guardarão os dados recolhidos, nem mesmo utilizar essa informação para outros fins, que não o da própria segurança interna. Porém, privilegiar o aumento da segurança nas ruas (que será uma clara vantagem) é, para os chineses, mais importante do que a desvantagem que apresenta.

3.5. Taxa de Afluência de Trânsito

A verificação da taxa de afluência de trânsito é feita com base na colocação de um conjunto de câmaras de videovigilância em pontos estratégicos dos eixos rodoviários, câmaras essas que permitem, de forma fidedigna, controlar o tipo de trânsito, a quantidade de veículos ou as horas mais críticas em termos de circulação… Com os dados por ele obtidos, podem, também, planear-se, com maior precisão, obras e mudanças ao trânsito, por exemplo, bem como, nalguns casos, aplicar coimas por manobras perigosas ou por excesso de velocidade, por exemplo.

3.6. API de direções

Para obter a rota ideal entre 2 pontos, é comum o recurso a serviços de APIs de direções. De entre todas as soluções disponíveis no mercado, destacam-se principalmente as APIs da Google Maps, Graphhopper e Mapbox Directions. No fundo, a utilização das mesmas depende da preferência pessoal pois todas oferecem vantagens semelhantes nos serviços, embora o Graphhopper tenha um modo offline a custo de guardar os mapas (ficheiros que ocupam bastante espaço) no dispositivo e a API da Google Maps permita retornar uma rota que evita trânsito

intenso, sendo até bastante semelhantes em termos de preço de serviços. Visto que a iClio já tinha contratado o serviço da Mapbox Directions, o mesmo foi utilizado para a criação deste projeto.

4. Tecnologias utilizadas

Para a criação da aplicação que aqui apresentamos, foram escolhidas diversas tecnologias a utilizar, que são descritas nesta secção do relatório.

Para uma melhor compreensão, serão já divididas as tecnologias utilizadas, de acordo com as diferentes componentes, anteriormente apresentadas.

4.1 Servidor

4.1.1 PostgreSQL

O PostgreSQL é um sistema de base de dados de objetos relacionais de projeto Open Source que se destacou de todos os outros ao implementar o controlo de concorrência multi versões, mantendo uma forma simples e prática na implementação de funcionalidades tais como os tipos de atributo definidos pelo utilizador, a herança de tabelas, vistas e regras, entre outros…

É atualmente um dos sistemas de base de dados relacionais mais utilizada no mundo.

Source: http://www.postgresqltutorial.com/what-is-postgresql/

4.1.2 Flask

A *Flask* é uma *microframework* do *Python* que permite a criação de servidores simplificada, com um servidor de desenvolvimento e um *debugger* embutidos, permitindo que o servidor possa seguir uma arquitetura *RESTful*.

Esta ferramenta é baseada no *Jinja 2*, uma linguagem de criação de *templates* para *python*, e no *Werkzeug*, uma biblioteca de utilidades *WSGI* (***W**eb **S**erver **G**ateway **I**nterface*), para garantir que é 100% compatível com as normas de *WSGI 1.0*. Trata-se, também, de uma ferramenta baseada em *unicode*.

A grande comunidade desta *framework* e a sua documentação extensiva tornam o uso e a explicação das funcionalidades da mesma muito mais simples, favorecendo o seu uso por parte dos utilizadores.

Para instalar esta *microframework*, é necessário utilizar o seguinte comando no terminal:

pip install Flask

Source: https://pt.wikipedia.org/wiki/Flask_(framework_web)

4.1.3 Flask-SQLALchemy

A *Flask-SQLAlchemy* é uma extensão do *flask* utilizada para permitir o suporte da aplicação *Flask* à *framework SQLAlchemy*.

O *SQLAlchemy* é um *ORM* (*Object Relational Mapper*) que permite a interação com a base de dados utilizada (*postgres*). Esta extensão permite a transcrição de dados Relacionais (da base de dados) em dados nativos de *python*.

Para instalar esta extensão, é necessário utilizar o seguinte comando no terminal:

pip install Flask-SQLAlchemy

Source: http://flask-sqlalchemy.pocoo.org/2.3/

4.1.4 Flask Marshmallow

A *Flask-Marshmallow* é uma extensão do *flask* que permite a conversão dos dados nativos em *python* para objetos que possam ser lidos pela *Frontend* (objetos *json*).

Trata-se de uma extensão de implementação bastante simples, que recorre à criação de esquemas pré-definidos para a desconstrução dos objetos em dados do tipo *python*, leitura e construção de dados do tipo *json*.

Para instalar esta extensão, deve utilizar-se o seguinte comando no terminal:

pip install flask-marshmallow

http://flask-marshmallow.readthedocs.io/en/latest/

4.1.5 Flask-CORS

A *Flask_CORS* (*Cross Origin Request System*) é uma extensão do *flask* que permite a pedidos entre diferentes origens (neste caso, trata-se do servidor e dos clientes de origens diferentes), com recurso à adição de informação, ao cabeçalho, do pedido *HTTP*.

Para instalar esta extensão, é necessário utilizar o seguinte comando no terminal:

pip install -U flask-cors

https://flask-cors.readthedocs.io/en/latest/

4.1.6 Postman

O *Postman* é uma ferramenta que facilita o teste de caminhos do servidor, permitindo a construção de um corpo e de um cabeçalho para um pedido (onde é enviada a informação do mesmo), a escolha do *URL* do pedido; finalmente, apresenta a resposta do servidor.

Esta é uma ferramenta ótima para o teste de todos os caminhos do servidor, antes mesmo de ter sido criado o suporte aos mesmos na parte do cliente (*frontend*).

O *software* deve ser instalado através do seguinte *link*:

https://www.getpostman.com/

4.1.7 VIRTUALENV

A *VIRTUALENV* é uma biblioteca *python* para criar um ambiente virtual de *python* numa diretoria, permitindo a instalação de bibliotecas / dependências *python* à vontade nessa mesma diretoria do projeto, não fazendo alterações no ambiente *python* original da máquina, o que, por sua vez, permite a criação de diversos ambientes diferentes *python* na mesma máquina, não criando, todavia, conflitos entre si.

Mesmo sendo o projeto desenvolvido o único a utilizar o ambiente *python*, é sempre "boa-prática" criar um ambiente virtual, no sentido de não interferir num possível projeto futuro, com um ambiente diferente.

Para instalar esta biblioteca, deve utilizar-se o seguinte comando no terminal:

pip install virtualenv

4.2 Modelos de classificação de densidade populacional

4.2.1 OpenCV2

A *OpenCV* (*Open Source Computer Vision Library*) é uma biblioteca *open source* bastante completa, que contém variadas funcionalidades, como por exemplo o processamento de:

- imagens e vídeo (x)
- *input* e o*utput* de vídeo (x)
- estruturas de dados
- álgebra linear
- algoritmos de visão computacional (filtros de imagem, reconhecimento de objetos, etc)(x)

Será importante referir que todas estas funcionalidades são executadas em tempo real.

(x) - funcionalidades utilizadas

Para instalar esta biblioteca, é necessário utilizar o seguinte comando no terminal:

pip install opencv-contrib-python

https://opencv.org/

4.2.2 Tensorflow

A *TensorFlow* é uma biblioteca *open source* de alta performance para aplicações de *machine learning*, sendo, neste caso, utilizadas as Redes Neuronais Convolucionais que a mesma possui, e que permitem o processamento de dados com recurso ao *CPU* ou à *GPU* do computador, consoante as necessidades de velocidade e os requisitos de *hardware* da máquina em que se criam os modelos de inteligência artificial (*CNN*s).

Para instalar esta biblioteca, deve utilizar-se o seguinte comando no terminal:

pip install --upgrade tensorflow

https://www.tensorflow.org/

4.3 Frontend

4.3.1 Node.js

Esta é uma linguagem de programação assíncrona e baseada em eventos, ideal para páginas dinâmicas, como as que existem no projeto desenvolvido ao longo deste estágio e aqui descrito.

O *Node.js* consiste numa plataforma destinada ao desenvolvimento de aplicações *server-side* sustentadas em rede, que utiliza o *JavaScript* e o *V8 JavaScript Engine*. Trata-se, pois, de uma plataforma com a qual podemos construir toda uma gama de aplicações *web*, recorrendo simplesmente ao código em *JavaScript*.

À primeira vista, pode não parecer tratar-se de uma informação particularmente interessante, já que há, atualmente, muitas outras formas de implementação deste tipo de aplicações. Porém, se nos debruçarmos um pouco mais sobre esta questão, veremos que, de acordo com a procura de aplicações na *internet* e pela forma como este código pode estruturar-se, existe todo um leque de novas possibilidades de desenvolvimento de aplicações *Web* e passaremos a ver este código como uma forte possibilidade de trabalho.

Importante será, neste ponto, referir que o Node.js pode ser *single threaded*: podendo apresentar-se, inicialmente, como um ponto fraco, ao trabalhá-lo verificamos que o uso deste código facilita bastante o desenvolvimento da aplicação, já que este Node.js usa uma abordagem não obstrutiva.

Relativamente ao *V8 JavaScript Engine*, é um interpretador aberto (chamado de open source). O facto de ter sido implementado pela *Google* em C++ e usado pelo *Chrome* parece provar as suas potencialidades e a sua validade e cria, naturalmente, uma grande expectativa relativamente ao seu desempenho.

Para utilizar esta linguagem de programação, é necessário obter o *software* no *link*:

https://nodejs.org/en/download/

https://nodejs.org/en/about/

4.3.2 VueJS

Trata-se de uma *Framework Open Source* de *JavaScript*, que permite a criação de *Interfaces* gráficas e de *frameworks* para *Web Applications*.

No projeto aqui descrito, esta *framework* foi utilizada para a criação da *Web Application*, com recurso ao *Webpack* disponível no *vue*, que permite a criação de uma página única para toda a *Web App* que será construída através de diversas componentes individuais, nomeadamente com recurso ao conceito de *routing* dos componentes num componente "Pai".

Para instalar esta *framework*, é necessário utilizar o seguinte comando no terminal, dentro da diretoria do projeto:

npm i vue

https://vuejs.org/

4.3.3 Mapbox GL JS

A *Mapbox GL JS* é uma biblioteca para a criação de mapas de vetores personalizados e interativos, que permite a importação de mapas com estilos pré-definidos ou facilmente personalizáveis e que funciona com base em pedidos feitos à *API* da *mapbox*, o que implica que o serviço de mapas, apesar de não funcionar em modo *offline*, permite o uso de mapas sem necessidade de os mesmos serem guardados em ficheiros *KML* ou em algum tipo semelhante.

Para instalar esta biblioteca, deve utilizar-se o seguinte comando no terminal, dentro da diretoria do projeto:

npm i mapbox-gl

https://www.mapbox.com/help/define-mapbox-gl-js/

4.3.4 Mapbox Directions API

Estamos perante uma *API* de complemento à biblioteca *Mapbox GL JS*, que permite a obtenção do caminho ideal, tendo em conta os mapas de estradas, bem como a sua distância e tempo de viagem. É utilizado para obter distâncias entre dois pontos de interesse, bem como para mostrar o caminho ideal entre os mesmos, desenhando-o no mapa e apresentando-o ao utilizador.

https://www.mapbox.com/help/how-directions-work/

4.3.5 jQuery

A *jQuery* é uma biblioteca *JavaScript* que tem como propósito a simplificação da escrita de código *JavaScript*, dando a possibilidade de que o programador utilize funcionalidades que permitam a manipulação código *HTML* / *CSS*, métodos *HTML*, pedidos *AJAX* e outras utilidades.

Devido à sua popularidade e à sua extensa documentação, tanto oficial como por parte da comunidade de utilizadores, esta ferramenta simplifica bastante a criação de código *JavaScript*.

Para instalar esta biblioteca, é necessário utilizar o seguinte comando no terminal na diretoria do projeto:

```
npm i jquery
```

https://jquery.com/

4.3.6 Bootstrap-vue

Trata-se de uma extensão à biblioteca *VueJS*, que permite a utilização de componentes e estilos pré-definidos na aplicação *Web App*.

O elevado número de componentes com diversos aspetos visuais permite a criação de páginas com um menor foco na componente estética do projeto, nomeadamente aquando duma fase de desenvolvimento, deixando as *Web Apps* com uma boa apresentação, ainda que de muito simples criação.

Para a utilização desta extensão, deve utilizar-se o seguinte comando no terminal da diretoria do projeto:

```
npm i bootstrap-vue
```

https://bootstrap-vue.js.org/

4.3.7 Vuejs-dialog

Consiste numa biblioteca de complemento ao *VueJS*, que permite a utilização de um "*prompt*" de *OK* / *Cancel* para operações diferenciadas, como a de apagar dados da base de dados, entre outras.

Esta biblioteca permite a criação de uma promessa muito mais simples para um finalizar / descartar dos dados obtidos, consoante a escolha do utilizador.

Para a utilização desta biblioteca, é necessário utilizar o seguinte comando no terminal na diretoria do projeto:

npm i vuejs-dialog

https://www.npmjs.com/package/vuejs-dialog

5 Servidor

Para uma melhor compreensão do processo de desenvolvimento e do funcionamento do servidor da aplicação desenvolvida, a apresentação do mesmo será dividida em 5 componentes, que a seguir se apresentam, de forma individual.

5.1 Criação do projeto

Como estrutura inicial do nosso projeto, deve ser criada apenas uma pasta com o nome do projeto desejado que, por sua vez, deve conter duas subdiretorias: uma designada por "*backend*" e outra designada por "*frontend*".

Após esta estrutura inicial estar definida, pode-se proceder à criação do servidor.

5.2 Criação do ambiente virtual

Para a criação do ambiente virtual, é necessário efetuar aos seguintes passos:

1. Dentro do terminal / linha de comandos, navegar até estar dentro da subdiretoria "*backend*";
2. Inserir o comando '*python3.6 -m venv "nome_do_env"*', para criar um ambiente virtual dentro da subdiretoria;
3. Navegar para a pasta *'/venv/bin/' com 'cd /venv/bin/' ;*
4. Inserir o comando '*source activate*' para ativar o ambiente virtual;
5. Navegar de volta para a subdiretoria "*backend*", inserindo o comando '*cd ..*' duas vezes.

Com o ambiente virtual criado e já ativo, nenhuma das mudanças efetuadas ao ambiente *python* realizadas afetará o ambiente *python* original da máquina, pelo que se pode proceder à criação do servidor.

5.3 Base de dados

A base de dados deste projeto foi criada em *Postgresql*, um *ORDBMS* (*Object-Relational DataBase Management System*).

Para a integração da base de dados *Postgresql* no servidor criado com recurso ao *Flask*, é necessária a utilização de uma extensão de *ORM* (*Object-Relational Mapping*). Como solução para este problema, é aconselhado o uso da extensão *Flask-SQLAlchemy*.

A utilização da *Flask-SQLAlchemy* permite a interação direta do servidor com a base de dados. Assim, podem ser utilizadas funções desta biblioteca no servidor para obter / criar / alterar / apagar dados da base de dados sem recurso a *queries* que, por vezes, podem ser complexas.

Para o projeto desenvolvido, foi necessário que a nossa base de dados guardasse informação relativa a cidades, categorias, pontos de interesse, câmaras, previsões, utilizadores e *tokens* de sessão, sendo que:

- Uma cidade será caracterizada pelo nome do país, da cidade, pelos valores de latitude e longitude do centro geográfico da cidade e pela matriz de distâncias associada à mesma;
- Uma categoria será caracterizada pelo seu nome e pelo nome do seu ícone;
- Um ponto de interesse terá uma cidade e uma categoria associadas, sendo que uma categoria e uma cidade podem estar presentes em diferentes pontos de interesse, mas um ponto de interesse apenas pode pertencer a uma categoria e a uma cidade, sendo possível registar informação do nome, da descrição, dos valores de latitude e longitude geográficas, do tempo médio de visita, da área e da prioridade do ponto de interesse, bem como dos identificadores únicos da cidade e da categoria associadas ao mesmo;
- Uma câmara terá o ponto de interesse a que está associada, sendo que um ponto de interesse pode ter várias câmaras, mas uma câmara só pode pertencer a um ponto de interesse, sendo possível registar informação do endereço *IP* da câmara e o identificador único do ponto de interesse associado ao mesmo;
- Uma previsão terá uma câmara associada, sendo que uma câmara poderá ter várias previsões, mas uma previsão apenas está associada a uma câmara, sendo possível registar informação sobre a classificação obtida pelo modelo a ser criado, bem como a data e hora da mesma;

- Um utilizador será caracterizado pelo seu *email*, *password* e pelo cargo que detém na aplicação em geral;
- Um *token* de sessão terá um utilizador associado, sendo que um utilizador poderá ter vários *tokens* associados, mas um *token* apenas está associado a um utilizador, sendo que um *token* é caracterizado pelo seu identificador único, que será o valor do *token* ao utilizar identificadores únicos do tipo *UUID*, pela sua data de criação e expiração e pelo identificador único do utilizador ao qual está associado.

Para as tabelas que guardam informação relativa às cidades, categorias, pontos de interesse, câmaras e *tokens*, deve, também, ser adicionado um campo designado de *active*, para ser implementado um *soft delete*; isto significa que os dados não são apagados, mas escondidos do utilizador, impedindo, assim, possíveis erros associados à remoção de registos de uma tabela de uma base de dados.

No caso da previsão, para a implementação do *soft delete* foi adicionado um campo designado de *latest* que, se estiver "verdadeiro", implica que a previsão é a mais recente para a câmara associada à mesma, sendo mudado para "falso" a cada inserção de uma nova previsão.

No caso do utilizador, não deverá ser implementado o *soft delete* e os registos deverão ser completamente removidos da base de dados a pedido do utilizador, de acordo com a RGPD.

Para corresponder a todos os requisitos do projeto desenvolvido anteriormente mencionados, apresenta-se, seguidamente, uma proposta para a estrutura da base de dados:

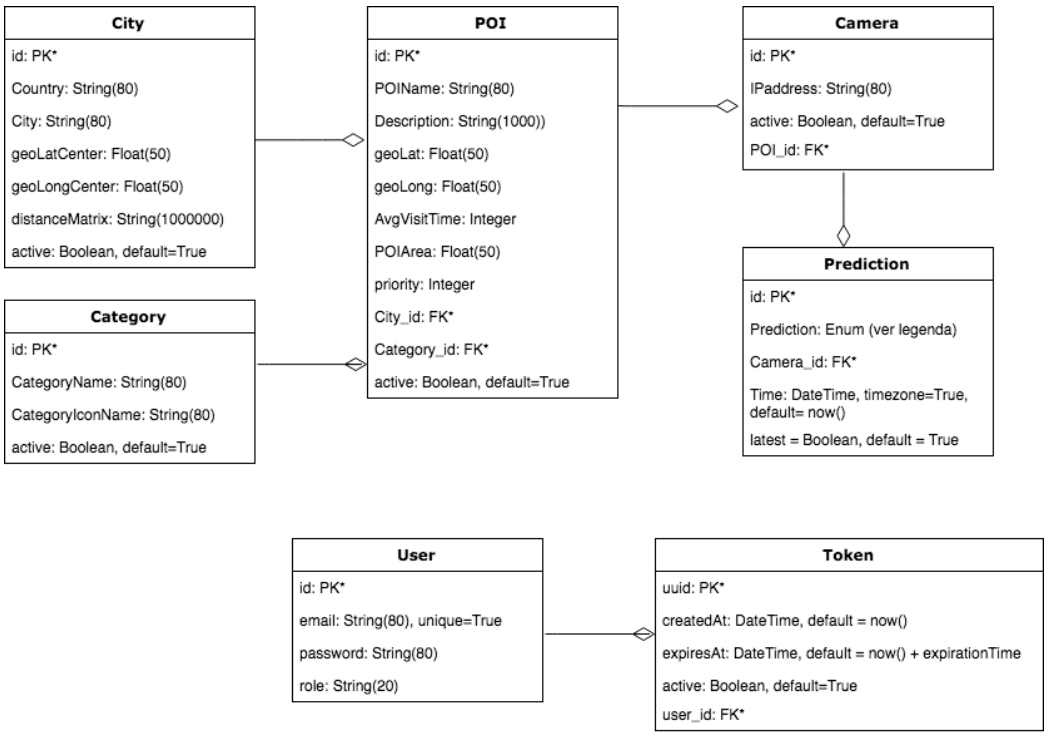


Figura 2 - Estrutura da base de dados

Para a criação da base de dados, é boa prática criar uma subdiretoria designada *models* (ou "modelos") na diretoria base do servidor onde vai ser criado o modelo de dados que representa cada tabela da mesma base de dados.

É, também, necessária a criação de um ficheiro chamado "*base.py*", que deverá ser importado em cada um dos modelos de dados criados, para permitir que o ficheiro *base.py* tenha acesso a toda a informação declarada nos ficheiros de modelos de dados.

Tendo em conta as tabelas apresentadas como necessárias ao funcionamento da aplicação na figura 2, a estrutura da diretoria e dos ficheiros a aplicar neste projeto é a que a seguir se apresenta:

Diretoria	**Ficheiro de modelo de dados**
Models	• base.py • city.py • category.py • poi.py • camera.py • prediction.py • user.py • token.py

Tabela 1 - Estrutura da diretoria relativa aos modelos do servidor

Dentro do ficheiro *base.py*, devem apenas ser criados um "*engine*" e uma sessão para a base de dados; é, ainda, necessário criar uma base declarativa para todas as tabelas que vão ser apresentadas de seguida, pelo que o ficheiro final apenas deverá conter o seguinte código:

```
1  from sqlalchemy import create_engine
2  from sqlalchemy.ext.declarative import declarative_base
3  from sqlalchemy.orm import sessionmaker
4  from flask_sqlalchemy import SQLAlchemy
5
6  engine = create_engine('postgresql://postgres@localhost/cdcv5')
7  Session = sessionmaker(bind=engine)
8
9  Base = declarative_base()
```

Figura 3 - Base declarativa para os modelos

Com a base declarada para o nosso modelo de dados, podem ser criadas as classes necessárias para o projeto (uma para cada tabela), em ficheiros individuais, nos quais devem ser importadas as bibliotecas do *SQLAlchemy* necessárias e se deve declarar uma classe que irá conter o objeto referente à nossa tabela na base de dados, dentro da qual, por sua vez, é necessário definir os seguintes aspetos:

- o nome da tabela da base de dados;
- os campos da base de dados (com os tipos, tamanhos, valores por defeito, entre outras características…)
- Duas funções para esta classe:
    - Função "*__init__*", que consiste numa função para a inicialização de uma nova entrada de dados, tendo em conta os parâmetros dessa mesma função;
    - Função "*__repr__*", que será uma função opcional para a representação de uma entrada de dados.

Em seguida, apresentam-se os diferentes modelos de dados criados para este projeto, para os quais se fará uma breve descrição dos mesmos.

5.3.1 Modelo *Category*

Nesta tabela serão guardados todos os atributos que caracterizam uma categoria de pontos de interesse. Trata-se de um aspeto fulcral na elaboração do projeto, já que permite estabelecer os pontos fundamentais que lhe servirão de base, com base as caraterísticas pretendidas para o *software* a desenvolver. Com efeito, não deixa de ser importante reiterar que o estabelecimento destas categorias representa o primeiro passo para que se possa avançar para o desenvolvimento do projeto. No caso que aqui se analisa, a lista escolhida para os mesmos é a que agora se apresenta:

Nome	**Tipo de dados**	**Descrição**
id	Inteiro, chave primaria	Identificador único da categoria de ponto de interesse
CategoryName	*String*(80)	Nome da categoria de ponto de interesse

CategoryIconName	*String*(80)	Nome do ícone a apresentar na visualização do ponto de interesse no mapa associado à categoria
active	*Booleano, default=True*	Indica se uma categoria está ativa ou não (isto é, se foi, entretanto, apagada)

Tabela 2 - Atributos da tabela "Category"

O código associado a este modelo pode ser encontrado no anexo 1.

5.3.2 Modelo *City*

Considerando que esta plataforma visa estar disponível, de forma fácil e inteligível, a qualquer utilizador que a queira usar, é fundamental começar por selecionar e indicar claramente os pontos de interesse, tornando-a agradável à vista e fácil de utilizar.

Sempre com esta prorrogativa em mente, nesta tabela serão guardados todos os atributos que caraterizam uma cidade, sendo a lista escolhida para os mesmos a que a seguir se apresenta:

Nome	**Tipo de dados**	**Descrição**
id	Inteiro, chave primaria	Identificador único da cidade
Country	*String*(80)	País a que pertence a cidade
City	*String*(80)	Nome da cidade
geoLatCenter	*Float*(50)	Valor de latitude a utilizar como centro do mapa da cidade
geoLongCenter	*Float*(50)	Valor de longitude a utilizar como centro do mapa da cidade
distanceMatrix	*String*(1000000)	Matriz de distâncias de todos os pontos de interesse associados a essa cidade, alterada a cada adição de um novo ponto de interesse. Recorre ao serviço *Mapbox Directions API* para obter a distância entre cada novo ponto e todos os pontos associados a essa cidade, que

		é baseado em pedidos *HTTP*. Ao manter uma cópia local da matriz de distâncias, é removido o tempo de espera que o utilizador teria para obter distâncias entre pontos no cálculo de rotas
active	*Booleano, default=True*	Indica se uma cidade está ativa ou não (isto é, se foi, entretanto, apagada)

Tabela 3 - Atributos da tabela "City"

O código associado a este modelo pode ser encontrado no anexo 2.

5.3.3 Modelo *POI*

À medida que avançamos na elaboração do projeto, a base de dados vai-se, evidentemente, completando, nomeadamente com as indicações que se pretendem juntar para melhorar a *performance* do *software* criado, mormente na ótica do utilizador, cuja vida se pretende facilitar.

Neste contexto, na tabela seguinte serão guardados todos os atributos que caracterizam um ponto de interesse, sendo a lista escolhida para os mesmos a que agora se apresenta:

Nome	**Tipo de dados**	**Descrição**
id	Inteiro, chave primaria	Identificador único do ponto de interesse
POIName	*String*(80)	Nome do ponto de interesse
Description	*String*(1000)	Descrição breve do ponto de interesse
geoLat	*Float*(50)	Valor para a latitude do ponto de interesse
geoLong	*Float*(50)	Valor para a longitude do ponto de interesse
AvgVisitTime	Inteiro	Tempo médio de visita, em minutos, de um ponto de interesse
POIArea	*Float*(50)	Área do ponto de interesse, em metros

priority	*Integer*	Valor de importância de visita associado ao ponto de interesse, valor entre [1;5]
active	*Booleano, default=True*	Indicação de se um ponto de interesse está ativo ou não (isto é, se foi, entretanto, apagado)
Category_id	*Integer*, chave estrangeira da tabela *Category*	Identificador único da categoria associada ao ponto de interesse
City_id	Inteiro, chave estrangeira da tabela *City*	Identificador único da cidade associada a um ponto de interesse

Tabela 4 - Atributos da tabela "POI"

O código associado a este modelo pode ser encontrado no anexo 3.

5.3.4 Modelo *Camera*

Na tabela que constitui o passo seguinte para o desenvolvimento do *software* serão guardados todos os atributos que caracterizam um ponto de interesse. Estes atributos vão sendo cada vez mais precisos, detalhados e concretos, visando a facilidade de utilização e a satisfação do utilizador, sendo a lista escolhida para os mesmos a que aqui se apresenta:

Nome	**Tipo de dados**	**Descrição**
id	Inteiro, chave primaria	Identificador único de uma câmara
IPaddress	*String*(80)	*String* de conexão para a câmara
active	*Booleano, default=True*	Indica se uma câmara está ativa ou não (isto é, se foi, entretanto, apagada)
City_id	Inteiro, chave estrangeira da tabela *POI*	Identificador único do ponto de interesse em que se encontra localizada a câmara

Tabela 5 - Atributos da tabela "Camera"

O código associado a este modelo pode ser encontrado no anexo 4.

5.3.5 Modelo *Prediction*

Nesta tabela serão guardados todos os atributos que caraterizam um ponto de interesse. Após grande reflexão, e tendo a vista a necessidade que o utilizador sente de não perder tempo desnecessário quando realiza a visita e a vontade de respeitar, por parte do programador, essa mesma necessidade, foi escolhida a lista para os mesmos, constante na tabela a seguir apresentada.

Nome	**Tipo de dados**	**Descrição**
id	Inteiro, chave primaria	Identificador único de uma previsão
Prediction	*Enum* – valores: • *Low_density* • *Medium_low_density* • *Medium_density* • *Medium_high_density* • *High_density*	Valor da previsão
latest	*Booleano, default=true*	Indicação de se determinada previsão é a mais recente ou não
Time	*Datetime*	Hora de criação da previsão
Camera_id	Inteiro, chave estrangeira da tabela *POI*	Identificador único da câmara a que está associada a previsão

Tabela 6 - Atributos da tabela "Prediction"

O código associado a este modelo pode ser encontrado no anexo 5.

É importante notar que a tabela *Prediction* é uma tabela que só permitirá ao utilizador, independentemente do seu "*role*", visualizar a informação da mesma, não lhe permitindo quaisquer alterações. Com efeito, para inserções e atualizações (a única atualização possível será o *soft delete* dos dados, alterando o atributo "*latest*" para o valor *false*. de dados, sendo que apenas os *scripts* dos modelos de classificação de ocupação de um ponto de interesse irão interagir com esta tabela.

5.3.6 Modelo *User*

Para o desenvolvimento e posterior aplicação deste projeto sob a forma de *software* acessível, é importante garantir não apenas a fiabilidade dos dados facultados, mas também a identidade do utilizador, pelo que o acesso à plataforma só será possível mediante uma inscrição prévia e um *login* a cada acesso. Desta forma, nesta tabela serão guardados todos os atributos que caracterizam um utilizador, sendo a lista escolhida para os mesmos a seguidamente apresentada:

Nome	**Tipo de dados**	**Descrição**
id	Inteiro, chave primaria	Identificador único de uma previsão
email	*String*(80)	*Email* ao qual o utilizador está associado
password	*String*(200)	*Password* escolhida pelo utilizador para autenticação. Para evitar que a *password* do utilizador seja guardada em texto simples, a mesma é encriptada através do método de *hashing* com *salt*.
role	*String*(20)	Indicação do tipo de utilizador, que pode ser ou utilizador normal ou administrador; é utilizado para gerir permissões do utilizador

Tabela 7 - Atributos da tabela "User"

O código associado a este modelo pode ser encontrado no anexo 6.

5.3.7 Modelo *Token*

Nesta tabela serão guardados todos os atributos que caracterizam uma sessão de um utilizador, sendo a lista escolhida para os mesmos apresentada de seguida:

Nome	**Tipo de dados**	**Descrição**
uuid	*UUID*, chave primaria	Identificador único de um *token* de sessão

createdAt	*Datetime, default = now()*	Hora de criação do *token* de sessão
expiresAt	*Datetime, default = now()* + prazo de expiração	Hora de expiração do *token* de sessão
active	*Booleano, default=True*	Indicação de se um *token* e válido ou não
User_id	Inteiro, chave estrangeira da tabela *POI*	Identificador único do utilizador a que está associado o *token* de sessão

Tabela 8 - Atributos da tabela "Token"

O código associado a este modelo pode ser encontrado no anexo 7.

5.3.8 Script DBManager.py

O *script DBManager.py* é um ficheiro para funções que impliquem qualquer inserção de dados na respetiva base.

Apesar de, por norma, este tipo de ficheiros de suporte não ser necessário, para interações com a base de dados mais complexas, como o sejam a inserção de um ponto de interesse ou mesmo a atualização da matriz de distâncias de uma cidade, que acontece a cada vez que um ponto de interesse é adicionado, ao calcular a distância desse ponto de interesse a todos os outros que estejam associados a essa mesma cidade.

As duas funções descritas são executadas em conjunto e o funcionamento das mesmas é apresentado na seguinte figura:

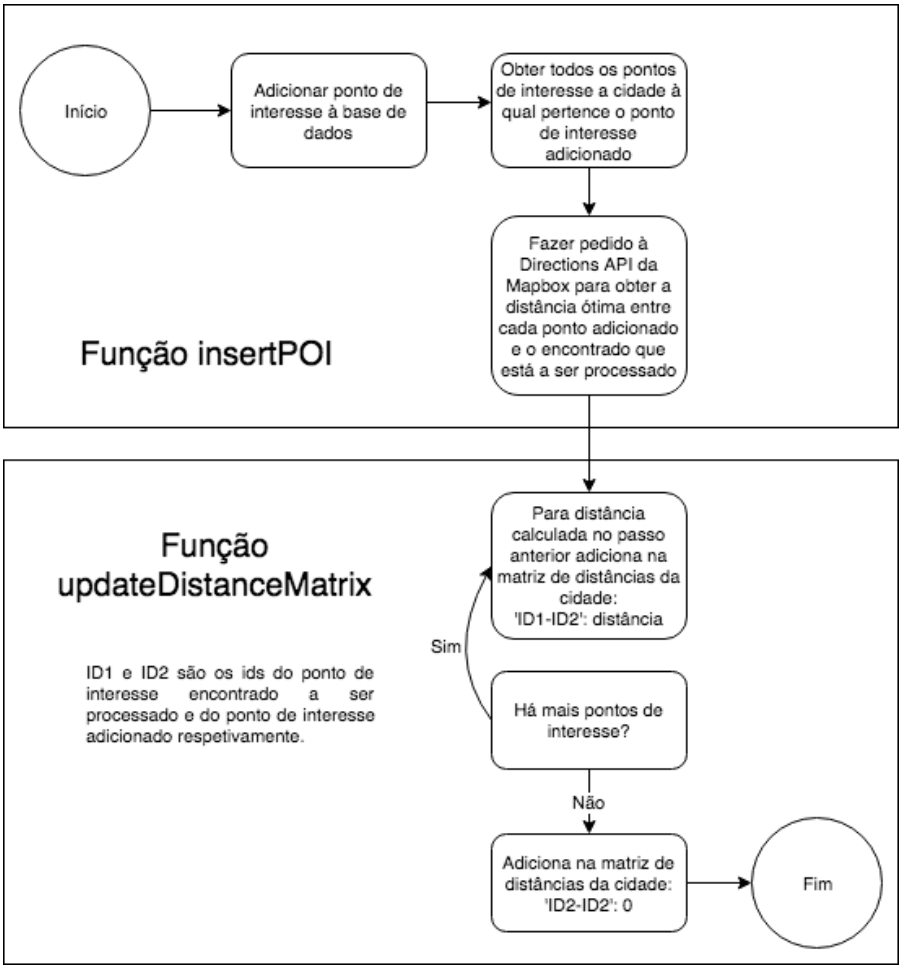


Figura 4 - Funcionamento do script DBManager.py

O código relativo a estas duas funções pode ser encontrado no anexo 8.

É, ainda, de notar que, visto que o já ficheiro existia neste projeto, foram também implementadas todas as inserções de dados na respetiva base, tal como se pode verificar no anexo 9.

5.4 RESTful API

REST (REpresentional State Transfer) é uma *API* (*Interface* de Aplicação de Programa) que utiliza pedidos *HTTP* para interagir com os recursos do servidor, baseando-se na tecnologia de transferência de estado representacional (*REST*).

Os recursos do servidor são acedidos por *URI*s (*Uniform Resource Identifiers*), em que cada *URI* se constitui como um pedido de interação com um determinado conjunto predefinido de recursos e os dados são enviados para o servidor / recebidos pelo cliente em formato, por norma, de dados *JSON*.

Tal como foi descrito anteriormente, as *RESTful API*s, baseiam-se em pedidos *HTTP*. Desta forma, parece-nos importante que, para uma melhor compreensão do processo de criação da *RESTful* necessária para este projeto, comecemos, numa primeira instância, por compreender a estrutura dos pedidos *HTTP*.

Comecemos, então, por assinalar que um pedido *HTTP* é constituído por:

- um *URI*, ou seja, um "caminho" para acesso à função no servidor e, por consequência, aos dados retornados pela mesma;
- um cabeçalho, no qual geralmente são enviadas informações do estado da aplicação, que podem ser precisos em diversas operações no servidor (por exemplo, o *token* de autenticação dado ao utilizador no *login*, a cada pedido que o utilizador faz ao servidor, deve ser enviado no cabeçalho do pedido e validado pelo servidor), ou seja, as características do pedido;
- um corpo, no qual são definidos os dados em sintaxe *JSON*, ou seja, a informação que o pedido contém.

Para cada pedido *HTTP* são, no fundo, criados dois objetos que englobam o *URI* com o qual se pretende comunicar (o cabeçalho e o corpo, tanto para o pedido de informação ao servidor, como para a resposta do servidor com a informação solicitada).

Os pedidos num conceito de uma *API RESTful* têm, apenas, uma pequena variância em comparação aos pedidos *HTTP*, que é a existência de verbos *HTTP*.

Ora os verbos *HTTP* são, de facto, palavras-chave para o tipo de acesso de informação que vem associado ao pedido, sendo que os verbos necessários para o projeto desenvolvido são:

- *GET*: serve para ler dados do servidor;
- *POST*: serve para adicionar ou enviar dados para o servidor;
- *PUT*: serve para atualizar ou substituir dados no servidor;

- *DELETE*: serve para apagar dados na base de dados, através do recurso ao servidor.

É verdade que existem outros tipos de verbos *HTTP*. No entanto, estes constituem todos os tipos de interação com a informação necessários para o desenvolvimento do projeto aqui descrito.

Para uma melhor compreensão do conceito de comunicação entre cliente e servidor num funcionamento em *RESTful API*, consideremos os seguintes pedidos como exemplo, mantendo sempre em vista que existe um servidor, que este está disponível e que está à espera de pedidos no endereço *http://localhost:8080*.

: URI	**Verbo HTTP**	**Tipo de acesso de informação**	**Exemplo**
http://localhost:8080/pois	GET	Obter informação	Obtém os dados relativos a pontos de interesse da base de dados
http://localhost:8080/pois	POST	Adicionar informação	Adiciona dados relativos a um ponto de interesse à base de dados
http://localhost:8080/pois	PUT	Alterar informação	Altera dados relativos a um ponto de interesse na base de dados
http://localhost:8080/pois	DELETE	Apagar informação	Apaga dados relativos a um ponto de interesse na base de dados

Tabela 9 - Exemplo de funcionamento de uma RESTful API

5.4.1 Caminhos do Servidor

O servidor terá como objetivo funcionar através do conceito de RESTful API, apresentado anteriormente, que é baseado em rotas (que, para evitar confusão com as rotas geradas pelos algoritmos e a apresentar ao utilizador, serão designadas, ao longo deste capítulo, como "caminhos do servidor") pré-definidas para o acesso e interação com os dados da base de dados.

Nas tabelas apresentadas de seguida, são explicadas as diferentes rotas criadas para o servidor, o seu objetivo e o fluxo dos dados associados às mesmas.

Apesar de todos os pedidos serem, também, constituídos por um cabeçalho, o mesmo será omitido nas tabelas de apresentação das rotas do servidor pois, em todos os pedidos, o cabeçalho

do mesmo é apenas constituído pelo *token* de sessão do utilizador (e o respetivo valor do *UUID*), gerado automaticamente, que é guardado nas *cookies* do navegador *web* para validação da autenticação do utilizador.

Pedidos com verbo HTTP GET:

Tabela(s) da Base de Dados	Rota	Descrição	Argumentos
POI	'/pois'	Obtém os dados relativos a todos os pontos de interesse ativos	N.D.
POI	'/pois/description'	Obtém os dados relativos ao ponto de interesse cujo *id* é enviado como argumento	*ID* do ponto de interesse do qual se pretendem obter os dados
POI	'/pois/findbyname'	Obtém os dados relativos ao ponto de interesse cujo nome é enviado como argumento	Nome do ponto de interesse do qual se pretendem obter os dados
POI	'/pois/filterby/cities'	Obtém os dados relativos a todos os pontos de interesse ativos que pertencem à cidade cujo *id* é enviado como argumento	*ID* da cidade da qual se pretende obter a lista de pontos de interesse associados
POI	'/pois/filterby/categories'	Obtém todos os pontos de interesse que pertencem à categoria cujo *id* é enviado como argumento	*ID* da categoria da qual se pretende obter a lista de pontos de interesse associados
Camera	'/cameras'	Obtém os dados relativos a todas as câmaras ativas	N.D.
Camera	'/cameras/description'	Obtém os dados relativos à câmara cujo *id* é enviado como argumento	*ID* da câmara da qual se pretende obter a informação
Camera	'/cameras/findbyip'	Obtém os dados relativos à câmara cujo endereço *IP* é enviado como argumento	Endereço *IP* da câmara da qual se pretende obter a informação
Prediction	'/predictions'	Obtém os dados relativos a todas as previsões mais recentes	N.D.
Prediction	'/predictions/description'	Obtém os dados relativos à previsão cujo *ID* é enviado como argumento	*ID* da previsão da qual se pretende obter a informação
City	'/cities'	Obtém os dados relativos a todas as cidades ativas	N.D.

City	'/cities/description'	Obtém os dados relativos à cidade cujo *ID* é enviado como argumento	*ID* da cidade da qual se pretende obter a informação
City	'/cities/findbyname'	Obtém os dados relativo à cidade cujo nome é enviado como argumento	Nome da cidade da qual se pretende obter a informação
City	'/cities/distanceMatrix'	Obtém a matriz de distâncias associada à cidade cujo *ID* é enviado como argumento	*ID* da cidade da qual se pretende obter a matriz de distâncias
Category	'/pois/categories'	Obtém os dados relativos a todas as categorias ativas	N.D.
Category	'/pois/categories/description'	Obtém os dados relativos à categoria cujo *ID* é enviado como argumento	*ID* da categoria da qual se pretende obter a informação
Category	'/pois/categories/findbyname'	Obtém todos os dados relativos à categoria cujo nome é enviado como argumento	Nome da categoria da qual se pretende obter a informação
User	'/register/uniqueEmail'	Verifica se já existe um utilizador com o *email* enviado como argumento	*Email* do utilizador que se pretende criar
User	'/users'	Obtém todos os dados relativos aos utilizadores ativos	N.D.
User	'/users/description'	Obtém todos os dados relativos ao utilizador cujo *ID* é enviado como argumento	*ID* do utilizador do qual se pretende obter a informação
User	'/users/findbyname'	Obtém todos os dados relativos ao utilizador cujo *email* é enviado como argumento	*Email* do utilizador do qual se pretende obter a informação
User	'/login/complete'	Obtém o *Token* de sessão atribuído ao utilizador, se o l*ogin* do mesmo for bem-sucedido	*Email* e *Password* do utilizador com o qual se pretende autenticar

Tabela 10 - Caminhos "GET" do servidor

O código associado a estes caminhos para o servidor pode ser encontrado no anexo 10.

Pedidos com verbo *HTTP POST*:

Tabela(s) da Base de Dados	Rota	Descrição	Dados no corpo do pedido *
POI	'/pois/add'	Adiciona, à base de dados, os dados relativos ao ponto de	• *POIName* • *Description*

		interesse, enviados no corpo do pedido	• *geoLat* • *GeoLong* • *AvgVisitTime* • *POIArea* • *priority* • *Category_id* • *City_id* • *POISFound* - lista de pontos de interesse associados à cidade cujo *ID* é definido antes, para os cálculos necessários à atualização da matriz de distâncias na tabela *City*
POI	'/pois/crowd_density'	Obtém a previsão associada à lista de pontos de interesse enviada no corpo do pedido	Lista de pontos de interesse da qual se pretende obter a previsão mais recente
Camera	'/cameras/add'	Adiciona, à base de dados, os dados relativos a uma câmara, enviados no corpo do pedido	• *IPaddress* • *POI_id*
Prediction	'/predictions/add'	Adiciona, à base de dados, os dados relativos a uma previsão, enviados no corpo do pedido	• Valor da previsão • *IPaddress* da câmara associada à previsão
City	'/cities/add'	Adiciona, à base de dados, os dados relativos a uma cidade, enviados no corpo do pedido	• *City* • *Country* • *geoLatCenter* • *geoLongCenter*
Category	'/categories/add'	Adiciona, à base de dados, os dados relativos a uma categoria, enviados no corpo do pedido	• *CategoryName* • *CategoryIconName*
User	'/*register*/*complete*'	Adiciona, à base de dados, um novo utilizador com os dados enviados no corpo do pedido e com o role "*user*"	• *Email* • *Password*
User	'/*register*/*admin*'	Adiciona, à base de dados, um novo utilizador com os dados enviados no corpo do pedido e com o *role* "*admin*"	• *Email* • *Password*

Tabela 11 - Caminhos "POST" do servidor

(*) - Informação a que diz respeito cada um dos atributos apresentada acima, nas tabelas de descrição de modelos de dados

O código associado a estes caminhos para o servidor pode ser encontrado no anexo 11.

Pedidos com verbo *HTTP PUT*:

Tabela(s) da Base de Dados	Rota	Descrição	Argumentos	Dados no corpo do pedido *
POI	'/pois/update'	Altera os dados relativos ao ponto de interesse cujo *id* é enviado como argumento, dados esses enviados no corpo do pedido, à base de dados	*ID* do ponto de interesse a alterar	• *POIName* • *Description* • *geoLat* • *GeoLong* • *AvgVisitTime* • *POIArea* • *priority* • *Category_id* • *City_id*
Camera	'/cameras/update'	Altera os dados relativos ao ponto de interesse cujo id é enviado como argumento, dados esses enviados no corpo do pedido, à base de dados	*ID* da câmara a alterar	• *IPaddress* • *POI_id*
Prediction	'/predictions/update'	Altera os dados relativos ao ponto de interesse cujo id é enviado como argumento, dados esses enviados no corpo do pedido, à base de dados	*ID* da previsão a alterar	• Valor da previsão • *IPaddress* da câmara associada à previsão
City	'/cities/update'	Altera os dados relativos ao ponto de interesse cujo id é enviado como argumento, dados esses enviados no corpo do pedido, à base de dados	*ID* da cidade a alterar	• *City* • *Country* • *geoLatCenter* • *geoLongCenter*

Category	'/categories/update'	Altera os dados relativos à categoria cujo id é enviado como argumento, dados esses enviados no corpo do pedido, à base de dados	*ID* da categoria a alterar	• *CategoryName* • *CategoryIconName*
User	'/users/update'	Altera o role do utilizador cujo id é enviado como argumento para o role enviado no corpo do pedido	*ID* do utilizador a alterar	• *Email* • *Password*

Tabela 12 - Caminhos "PUT" do servidor

(*) - Informação a que diz respeito cada um dos atributos apresentada acima, nas tabelas de descrição de modelos de dados

O código associado a estes caminhos para o servidor pode ser encontrado no anexo 12.

Pedidos com verbo *HTTP DELETE*:

Para evitar possíveis complicações ao apagar dados da respetiva base, é considerado "boa prática" criar um *soft delete* dos registos na mesma base de dados.

Para criar este *soft delete*, apenas é necessário passar um atributo designado de "*Active*" para "verdadeiro" ou "falso". Assim, apesar de não serem apagados dados nenhuns da respetiva base, os mesmos podem ter o acesso bloqueado para o utilizador (o que leva ao mesmo resultado do que apagar os mesmos, evitando, porém, possíveis conflitos de relações entre tabelas e dados).

Apesar do *soft delete* ser visto como uma boa prática, de acordo com o *RGPD*, dados pessoais e relativos às contas têm que ser, efetivamente, apagados completamente a pedido do utilizador. Num mundo cada vez mais global, onde cada vez temos mais cartões desta ou daquela empresa e onde, simultaneamente, cada vez nos sentimos mais dependentes da informática, um mundo onde, frequentemente, nos exigem dados pessoais para nos "oferecerem" esta ou aquela vantagem, a política legal de proteção de dados atualmente vigente garante, ao utilizador, a

privacidade dos dados que insere em qualquer plataforma, seja ela informática / virtual ou não. Daí decorre esta obrigatoriedade de, em qualquer momento, apagar todos os dados por solicitação do utilizador, uma vontade que deve ser estritamente respeitada e, consequentemente, cumprida. Esta questão da política de proteção de dados pode, aliás, ser consultada no *link* da Comissão Europeia https://ec.europa.eu/info/law/law-topic/data-protection/reform_pt, que nos permite ter a noção exata das implicações desta legislação.

Tabela(s) da Base de Dados	Rota	Descrição	Argumentos
POI	'/pois/delete'	Altera o atributo booleano "*active*" associado ao ponto de interesse cujo *ID* é enviado como argumento para *False*	*ID* do ponto de interesse a apagar
Camera	'/cameras/delete'	Altera o atributo booleano "*active*" associado à câmara cujo *ID* é enviado como argumento para *False*	*ID* da câmara a apagar
Prediction	'/predictions/delete'	Altera o atributo booleano "*latest*" associado à previsão cujo *ID* é enviado como argumento para *False*	*ID* da previsão a apagar
City	'/cities/delete'	Altera o atributo booleano "*active*" associado à cidade cujo *ID* é enviado como argumento para *False*	*ID* da cidade a apagar
Category	'/categories/delete'	Altera o atributo booleano "*active*" associado à categoria cujo *ID* é enviado como argumento para *False*	*ID* da categoria a apagar
User	'/users/delete'	Apaga completamente os dados associados ao utilizador cujo ID é enviado como argumento	*ID* do utilizador a apagar
Token	'/logout'	Altera o atributo booleano "*active*" associado ao *token* de sessão cujo *UUID* é enviado como argumento	*UUID* do *token* de sessão a apagar

Tabela 13 - Caminhos "DELETE" do servidor

(*) - Informação a que diz respeito cada um dos atributos apresentada acima, nas tabelas de descrição de modelos de dados.

O código associado a estes caminhos para o servidor pode ser encontrado no anexo 13.

É de notar, também, que foi criada uma função interna do servidor para a validação de *tokens*, que não faz parte do conceito de *API RESTful*, e que, sendo uma função interna do servidor, não implica a criação de um pedido, pelo que não possui nem verbo *HTTP*, nem cabeçalho, nem corpo.

Esta é uma simples função que apenas verifica se o *token* parâmetro existe na base de dados e se está ativo, retornando a indicação de "verdadeiro" ou "falso" consoante o resultado, tal como se pode verificar na figura seguinte:

```
#validate a token (for all operations)
def validateToken(tokeninfo):
    userToken = session.query(token).filter(and_(token.uuid == tokeninfo),(datetime.datetime.now() < token.expiresAt)).first()
    if(userToken):
        return True
    else:
        return False
```

Figura 5 - Função de validação de token de acesso

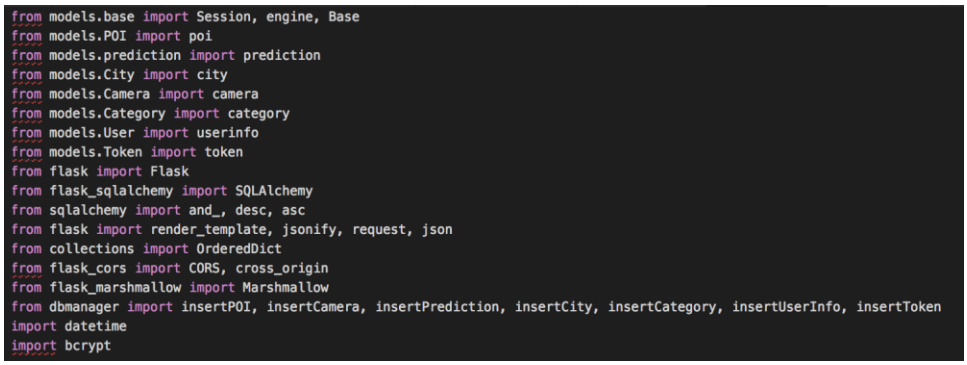
5.5 Criação e execução do servidor

A criação de um servidor é um processo bastante simples, quando feito com recurso à biblioteca *Flask*.

No terminal, dentro da diretoria "*backend*" criada anteriormente, devem ser instaladas as diversas bibliotecas a utilizar no ambiente virtual, com recurso ao gestor de pacotes *python PIP*.

Quando o ambiente virtual estiver configurado, deve ser criado um ficheiro com o nome "*index.py*", que será o ficheiro que contém todo o código do servidor.

De seguida, é necessário que todas as bibliotecas a utilizar, modelos de dados criados e o ficheiro "*DBManager.py*", descritos anteriormente, sejam importadas para o nosso servidor, como apresentado na seguinte figura:



```
from models.base import Session, engine, Base
from models.POI import poi
from models.prediction import prediction
from models.City import city
from models.Camera import camera
from models.Category import category
from models.User import userinfo
from models.Token import token
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import and_, desc, asc
from flask import render_template, jsonify, request, json
from collections import OrderedDict
from flask_cors import CORS, cross_origin
from flask_marshmallow import Marshmallow
from dbmanager import insertPOI, insertCamera, insertPrediction, insertCity, insertCategory, insertUserInfo, insertToken
import datetime
import bcrypt
```

Figura 6 - Bibliotecas e ficheiros importados para o servidor

Em termos de configuração do servidor, basta apenas definir um objeto do tipo *app* e definir:

- *String* de conexão à base de dados;
- Para desenvolvimento, deve ser ativado o modo *debug*. No entanto, em produção, esta configuração deve ser retirada para impedir que o servidor descreva os erros que possam existir a utilizadores mal-intencionados;
- Com recurso à biblioteca *Flask-CORS*, permitir pedidos de "*cross-origin*" e cabeçalhos necessários ao funcionamento da aplicação;
- Iniciar uma sessão de conexão à base de dados.

Na seguinte figura apresenta-se o código necessário para aplicar a configuração descrita:

```
#definicao da app
app=Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI']='postgresql://postgres@localhost/cdcv5'
app.debug=True
CORS(app, resources=r'/*', allow_headers=['token', 'Origin', 'Content-Type', 'Accept', 'Authorization', 'X-Request-With'], supports_credentials=True)

#abre sessao para coneccao a BD
session = Session()
#criacao de objeto marshmallow
ma=Marshmallow(app)
```

Figura 7 - Código necessário para a criação de um servidor vazio

5.6 Esquemas para os modelos de dados

Neste ponto, já temos os modelos de dados criados e o servidor funcional. No entanto, os dados que são guardados na base de dados são guardados como dados do tipo *Object-Relational*, que não pode ser interpretado pelos tipos de dados suportados pelo *python*.

Como solução para este problema, deve ser utilizada a extensão da biblioteca *Flask*, a *Flask-Marshmallow*, que é um *Object-Relational Mapper* e serve para mapear dados nativos de *python* para dados relacionais, com recurso a esquemas previamente definidos.

Os esquemas *marshmallow* mencionados têm que ser criados para cada modelo de dados (isto é, para cada tabela), seguindo sempre a mesma estrutura.

O exemplo do esquema criado para o modelo de dados associado a cada ponto de interesse é o que a seguir se apresenta:

```
#schemas para conversao de dados para json sem erros
class POISchema(ma.Schema):
    class Meta:
        # Fields to expose
        fields = ('id', 'POIName', 'Description', 'geoLat', 'geoLong', 'AvgVisitTime', 'POIArea', 'priority', 'active', 'Category_id', 'City_id')

poi_schema = POISchema()
poi_schema = POISchema(many=True)
```

Figura 8 - Exemplo de Schema para o modelo POI

Tal como se pode verificar na figura anterior, é muito simples a implementação de um esquema de dados *marshmallow*: basta apenas criar uma classe com o nome da tabela do modelo de dados, definir os campos que pertencem a cada registo e permitir que possam ser processados registos, um a um, ou vários de uma vez.

O mesmo processo de criação do esquema de dados tem que ser replicado para todas as tabelas da base de dados, tal como se pode verificar no anexo 14.

6 Modelos e algoritmos de rotas

Tendo em consideração que o objetivo final do projeto é permitir ao utilizador criar rotas personalizadas de forma a evitar pontos de interesse sobrelotados, foi necessária a criação de um classificador de densidade populacional num determinado ponto de interesse.

Para conseguir classificar a densidade populacional foi, inicialmente, decidido o uso de imagens de satélite como fontes de dados, com o objetivo de utilizar reconhecimento de imagem para o tratamento e classificação dos dados.

6.1 Aprendizagem automática

A aprendizagem automática é o processo de descoberta de conhecimento em bases de dados, ou seja, o processo de identificação de dados possivelmente úteis para a extração de conhecimento através da descoberta de padrões.

Este tipo de aprendizagem pode ser distinguido em dois tipos diferentes, sendo os mesmos a supervisionada e a não supervisionada.

A aprendizagem supervisionada é o tipo de aprendizagem baseado em exemplos, ou seja, tal como o nome indica, o analista terá um papel importante no processo de aprendizagem ao supervisionar o tratamento de dados e classificar manualmente as imagens do dataset de treino.

Por outro lado, a aprendizagem não supervisionada é o tipo de aprendizagem baseado na observação e descoberta de padrões genéricos nos dados do caso de estudo.

Dentro da aprendizagem supervisionada, existem tipos principais de aprendizagem, sendo os mesmos a classificação e a regressão de dados.

A regressão de dados baseia-se na previsão de um valor numérico e é, por norma, utilizada para estudos de mercado e projeções financeiras, entre outras aplicações. Poderia ser aplicada para o projeto aqui descrito, mormente para uma criar um modelo de aprendizagem automática que retornasse um valor numérico da taxa de ocupação do vídeo obtido pela câmara.

Por outro lado, a classificação dos dados baseia-se na divisão dos dados entre classes que indicam os possíveis resultados finais. Este será o tipo de aprendizagem automática ideal a

implementar no caso de estudo aqui descrito, com o intuito de dividir os dados entre classes que descrevam diversos níveis de taxa de ocupação.

Para um melhor estudo dos dados, é boa prática a divisão do *dataset* inicial em dois *datasets*: *o dataset* de treino, que corresponde, por norma, a 70% dos dados do *dataset* inicial que são utilizados no treino do modelo, e o *dataset* de teste, que corresponde aos restantes 30% dos dados e que são utilizados para o teste do modelo e para a obtenção das métricas de performance associadas ao mesmo, nomeadamente a Precisão de Classificação e a Perda de Informação, para um estudo do sucesso da utilização do modelo criado no caso de estudo.

6.2 Reconhecimento de imagem

Para a utilização de imagens de satélite como fonte de dados para a classificação da taxa de ocupação de um ponto de interesse, foram explorados ferramentas e métodos de reconhecimento de imagem e foi decidido o uso da biblioteca *OpenCV2* para tratamento de imagem e da biblioteca *TensorFlow* para a aplicação dos métodos de reconhecimento de imagem.

Como método de reconhecimento de imagem para este projeto, foi decidida a utilização de uma Rede Neuronal Convolucional, por se tratar de um método que, apesar de ter um processamento associado demorado e com grande escalabilidade temporal no treino e carregamento do modelo, de acordo com a quantidade de dados de entrada, apresenta, também, o melhor desempenho, na maioria dos casos, do que qualquer outro método existente.

Quanto às Redes Neuronais Convolucionais (CNN), é importante referir que se traduzem num método de reconhecimento de imagem que se inspira na biologia do córtex visual do olho humano.

Apesar de ser baseada nas redes neuronais tradicionais, esta foi uma ideia que surgiu após uma experiência levada a cabo, em 1962, por Hubel e Wiesel, na qual os mesmos conseguiram verificar que, dentro do córtex visual, existiam pequenos grupos de células que apenas se ativavam na presença de certas bordas de objetos e na sua orientação, tendo concluído que algumas células ficam ativas apenas com bordas horizontais, enquanto outras apenas ficam ativas com bordas verticais.

Este conceito de que células ou, no caso das redes neuronais, neurões, podem procurar e ativar-se de uma forma mais especializada levou ao aparecimento deste conceito de CNN.

As CNNs são, pois, um método que recebe uma imagem e que, ao aplicar-lhe diversas camadas convolucionais, de diminuição de dados e de "*fully-connected*", levam à obtenção de uma classificação final.

Na figura seguinte podemos encontrar um esquema de funcionamento de uma CNN:

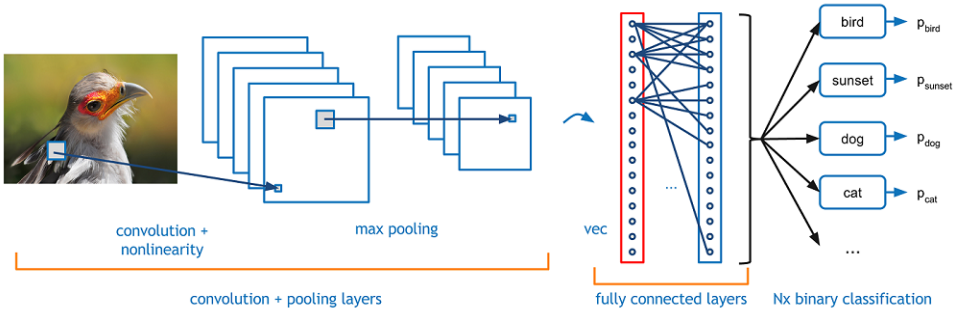


Figura 9 - Funcionamento de uma Rede Neuronal Convolucional

(imagem retirada de https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/)

Porém, as camadas Convolucionais decompõem uma imagem para processamento, mas não o fazem da mesma forma que o olho humano. De facto, o olho humano tem a capacidade de interpretar a cor de um objeto através daquilo que, na computação é, muitas vezes, decomposto em 3 dimensões (*RGB – Red, Green, Blue*), o que pode revelar-se um problema que, é, no entanto, de relativamente fácil resolução.

Com efeito, como solução para este problema, é aplicada uma camada Convolucional à imagem, para a qual devem ser definidos quatro parâmetros iniciais: o filtro a aplicar, o *padding*, o *stride* e o tipo de ativação a aplicar nessa mesma camada.

Ao serem definidos estes parâmetros, a imagem é decomposta em componentes com a dimensão do filtro a aplicar, repetindo este processo ao longo de cada coluna, avançando sempre o número de colunas igual ao valor de *padding* definido, e ao longo de cada linha, avançando sempre o número de linhas igual ao valor de *stride* definido.

Tendo em conta que o modelo criado servirá como prova de conceito, os métodos de ativação utilizados foram os que foram definidos como padrão pelas fontes de dados acedidas aquando a investigação destes modelos.

A aplicação do filtro da Convolução, cujo processo seria repetido avançando X a X colunas e Y a Y linhas, em que X seria o valor de *padding* definido e Y seria o valor de *stride* definido, aparece demonstrada na figura seguinte:

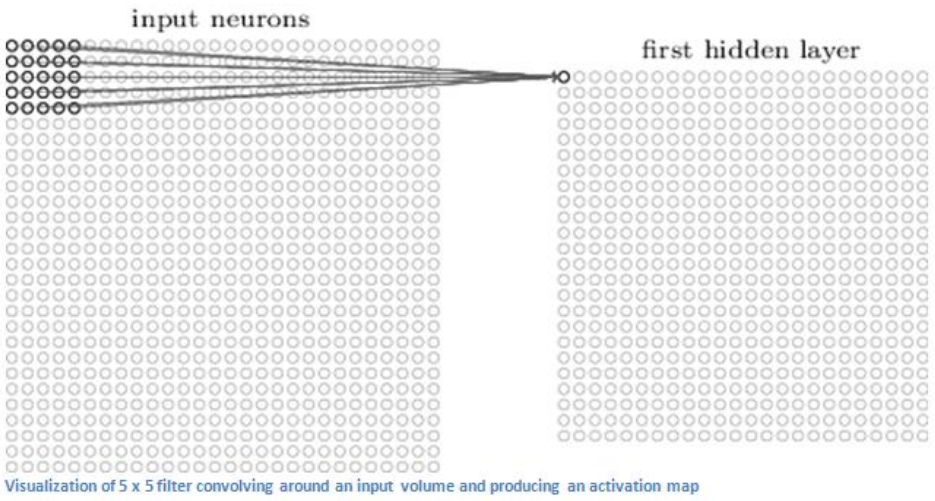


Figura 10 - Exemplo de um processo de convolução

É de notar que a "*first hidden layer*" resultante teria, para cada célula de posição [i,j], uma terceira dimensão associada, que conteria os valores de *Red*, *Green* e *Blue* que representam a cor.

Criada a camada Convolucional, procede-se à aplicação de uma camada de diminuição de volume de dados, designada de "camada de *Max Pooling*".

Na camada de *Max Pooling*, é aplicado um novo filtro à camada resultante da convolução, sendo o tamanho do filtro definido de novo como parâmetro. Aqui, a imagem é, de novo, decomposta em fragmentos de dimensão do filtro definido e é guardado o valor máximo dentro desse fragmento, construindo o resultado da camada com os valores guardados dos máximos de cada fragmento.

A aplicação desta camada tem como objetivo garantir que o resultado é uma matriz em que diferenças pequenas entre valores de cor entre pixéis vizinhos serão diminuídas e diferenças grandes entre valores de cor entre pixéis vizinhos serão aumentadas.

Estas duas camadas descritas anteriormente podem ser aplicadas em sucessão, para levar ao melhoramento do resultado das mesmas.

Com a aplicação das camadas Convolucional e de *Max Pooling*, está completo o processamento de dados e pode proceder-se à parte da Rede Neuronal que trata da análise e classificação dos dados.

Para isto, é necessária a criação de uma camada "*Fully Connected*", que é uma camada que cria um mapa de ligação entre os resultados expectáveis e os grupos de neurónios dos dados de entrada que estão ativos.

Para melhorar a performance das camadas "*Fully Connected*", é, por norma, aplicado um método simples de *dropout*, em que uma percentagem de neurónios definida como parâmetro é escolhida aleatoriamente e descartada depois, para forçar os restantes neurónios a melhorar a sua performance.

Por norma, são definidas, pelo menos, duas camadas "*Fully Connected*", separadas por um *dropout*, em que a primeira tem uma quantidade elevada de neurónios (geralmente 512 /1024/ 2048), sendo que a última terá o número de neurónios igual ao número de classes existente para o problema de classificação.

Na figura seguinte é apresentado o funcionamento das camadas de *fully connected* (à esquerda) e da camada de *dropout* (à direita):

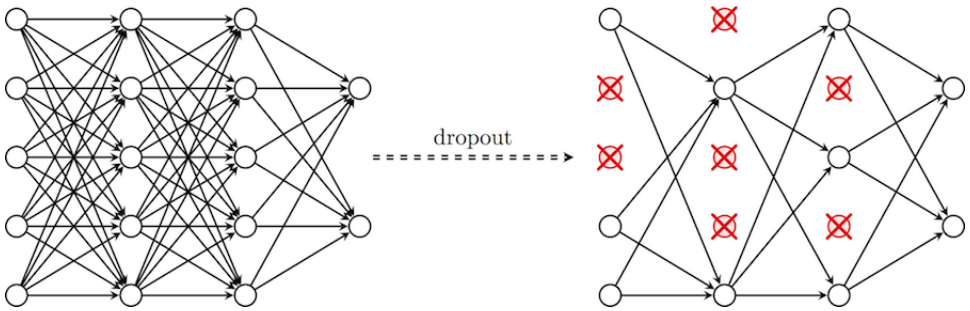


Figura 11 - Exemplo de um Dropout

Para uma melhor compreensão do estudo dos dados, do processo de criação de um classificador baseado em reconhecimento de imagem e das boas práticas na criação do mesmo, foram realizados dois principais testes, que a seguir descreveremos.

6.2.1 Imagens de números escritos à mão

Este primeiro teste consistiu no reconhecimento de imagens de números de 0 a 9 escritos à mão e o *dataset* necessário para a implementação do mesmo pode ser encontrado no *link* http://yann.lecun.com/exdb/mnist/.

Exemplos das imagens neste *dataset* podem ser visualizados na seguinte figura:

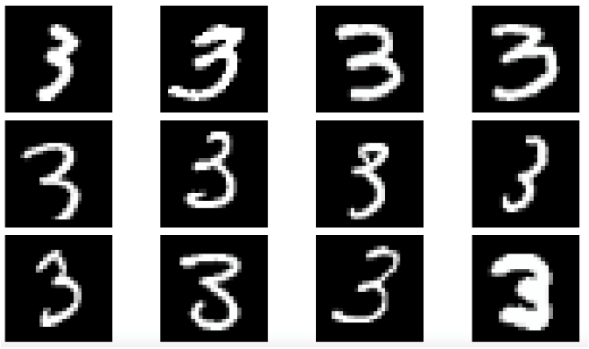


Figura 12 - Exemplos das imagens do dataset de números escritos à mão

Para o desenvolvimento deste classificador, foi necessária a criação de 3 *scripts* (ou módulos), sendo estes os seguintes:

- *Deepneuralnet.py*
- *Train.py*
- *Prediction.py*

No módulo *Deepneuralnet.py* será implementada a Rede Neuronal Convolucional.

A Rede Neuronal Convolucional, neste teste, será constituída por :

- Entrada de imagens com dimensão [28,28,1], ou seja, 28 pixéis de altura por 28 pixéis de largura, com apenas 1 dimensão de valor de cor;
- Camada de Convolução com um filtro de 8x8, *padding* de 3 e *stride* de 1, e ativação *ReLu*;
- Camada de *Max Pooling* com um filtro de 2x2 e 2 de *stride*;
- Camada de Convolução com um filtro de 8x8, *padding* de 3 e *stride* de 1, e ativação *ReLu*;
- Camada de *Max Pooling* com um filtro de 2x2 e 2 de *stride*;
- Camada de *Fully Connected* com 1024 neurónios e ativação *TanH*;
- Camada de *dropout* de 50% dos neurónios;
- Camada de *Fully Connected* com 10 neurónios (um por cada classe existente no *dataset*) e ativação *softmax*.

O código deste módulo pode ser visualizado no anexo 15.

No módulo *Train.py*, deve ser importada a Rede Neuronal Convolucional criada e deve ser declarado o treino do modelo, definindo, como parâmetros, o número de iterações a executar (refira-se que, quanto mais iterações houver, maior será o custo de tempo, mas também maior

será a precisão do modelo treinado) e é definida a medida de precisão de classificação como métrica para o treino.

É, também, essencial que seja definido, no módulo *Train.py*, um *validation dataset*, com o intuito de diminuir a situação de *Overfitting* (classificação muito precisa dos dados do *dataset*, mas incapacidade de classificar qualquer imagem que não esteja dentro do *dataset* de treino), que, por norma, é definido como correspondendo a 30% dos dados do *dataset* de treino.

O código deste módulo pode ser visualizado no anexo 16.

No módulo *Prediction.py*, é apenas carregado o modelo para memória, aplicado a uma imagem, e é retornado o resultado da classificação obtido na aplicação da Rede Neuronal Convolucional.

O código deste módulo pode ser visualizado no anexo 17.

6.2.2 Imagens do *dataset CIFAR 10*

A apresentação deste *dataset*, conhecido mundialmente, pode ser visualizada na figura seguinte:

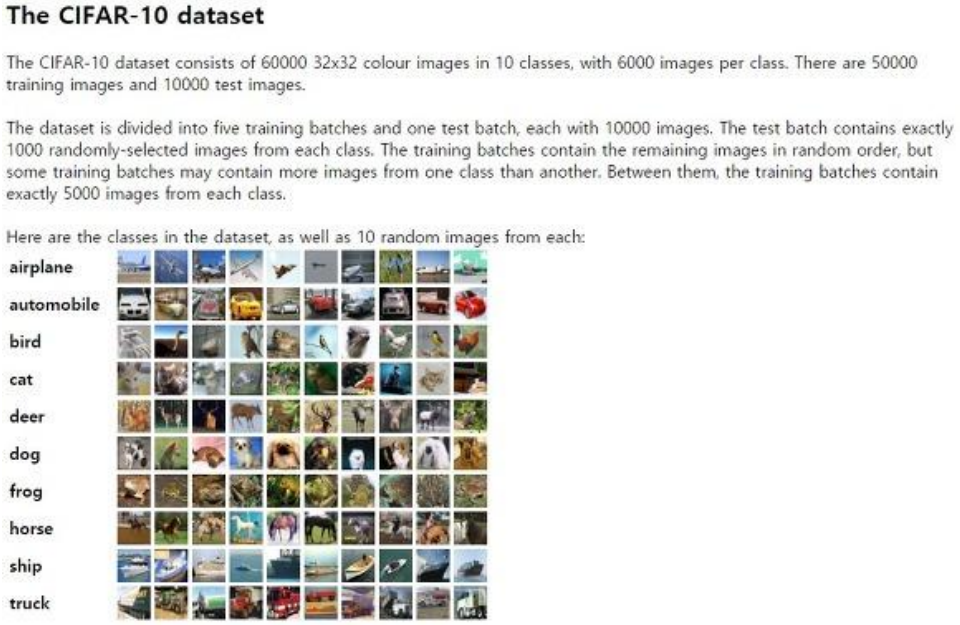


Figura 13 - Descrição do dataset CIFAR-10

Este teste também apresenta os mesmos três módulos de *DeepNeuralNet.py*, *Train.py* e *Prediction.py*, que são semelhantes aos módulos explicados para o teste de reconhecimento de imagens de números escritos à mão.

No entanto, destacam-se a existência de imagens a 3 dimensões de cor neste *dataset* (*Red*, *Green* e *Blue*) e a implementação de técnicas de pré-processamento e de aumento de dados.

No que diz respeito à implementação de técnicas de pré-processamento e aumento de dados, foram aplicadas apenas rotações e aumentos às imagens de forma a aumentar a dimensão do *dataset* e, consequentemente, a precisão do classificador final.

Foi, também, criado um teste de reconhecimento de imagens de cães e gatos, que apenas se encontra em anexo, devido às suas claras semelhanças com os dois testes descritos anteriormente.

Este projeto pode ser visualizado nos anexos 18 (código da rede neuronal convolucional criada), 19 (código do script para o treino da rede neuronal convolucional) e 20 (código do script para previsão de novas imagens com recurso à rede neuronal convolucional criada com o código do anexo 19) .

Após estes 3 testes estarem completos, procedeu-se à recolha de imagens de satélite.

6.2.3 AOI Image Exporter

Já com conhecimentos base de reconhecimento de imagem, procedeu-se à criação de um *script* de recolha de imagens de satélite, com base no serviço *WebMapService*, da *OWSLIB*.

A implementação deste serviço foi feita num *script* em que era inserida uma *string* de conexão de *WMS*, obtida através de uma plataforma da *EOS*, onde se definem as coordenadas de georreferência da área de interesse e se obtém, como resultado, esta *string* de conexão. Assim,

pode ser implementado o script onde se exporta a imagem obtida, que aqui é apresentado:

```
from owslib.wms import WebMapService

#change the link to the AOI WMS connection URL

URL=raw_input('Input the wms connection URL:\n')
wms = WebMapService(URL,version='1.1.1')

def getImage():
    BBWGS84=wms['TRUE_COLOR'].boundingBoxWGS84#get the Bounding Box in WGS84
    crsOpt= 'EPSG:4326'#set srs
    InputSize = (1024,1024)#set image size
    #get image
    img = wms.getmap(layers=['TRUE_COLOR'],srs=crsOpt,bbox=BBWGS84,size=InputSize,format='image/jpeg',transparent=True)
    return img

def exportImage(img):
    out=open('AOIImage.jpg','wb')
    out.write(img.read())
    out.close()

AOI=getImage()
exportImage(AOI)
```

Figura 14 - Código do AOIImageExporter

Apesar de este teste ter sido concluído com sucesso e de ser possível obter as imagens de satélite das áreas de interesse, foi possível verificar, nas imagens recolhidas, que as mesmas não apresentavam resolução suficiente para a identificação de objetos de pequena dimensão (pessoas ou qualquer outro tipo de objetos que pudesse ocupar um espaço mais restrito de um ponto de interesse).

Como tal, foram iniciadas negociações com a entidade responsável pela obtenção das imagens de satélite. No entanto, ficou claro que a maior resolução que poderia ser obtida não nunca iria ser o suficiente para as necessidades do projeto.

Após alguma consideração, foi decidido mudar a fonte de dados para o classificador para vídeos de câmaras de videovigilância, tendo como funcionamento ideal a recolha de pequenos vídeos que, para cumprimento do RGPD (Regulamento Geral da Proteção de Dados), seriam apenas representações instantâneas do movimento entre duas *frames*, impedindo que pudessem ser reconhecidas quaisquer características identificativas das pessoas filmadas e apagando-se, posteriormente, todos os vídeos e imagens obtidas pelo classificador, com o intuito de se manter apenas um valor de classificação final.

6.3 Reconhecimento de vídeo

Para a criação de modelos de aprendizagem supervisionada, foi escolhida a linguagem *python* que, com recurso às bibliotecas *CV2* (tratamento de vídeo) e *tensorflow* (criação de modelos de aprendizagem supervisionada), permite a criação de uma aplicação bastante simplificada de modelos complexos de aprendizagem.

Para reconhecimento de vídeo, optou-se por criar um modelo baseado numa rede neuronal que, ao ser treinado com um conjunto de dados reais composto por diversos pontos de interesse, com diferentes taxas de ocupação e diferentes condições atmosféricas, irá, mais tarde, receber um excerto de vídeo de uma câmara fixa num ponto de interesse e retornar, apenas, a sua classificação para a taxa de ocupação do mesmo.

No entanto, um dos grandes problemas do reconhecimento de vídeo em sítios bastante movimentados é a enorme quantidade de cores, formas e tamanhos possíveis para os nossos objetos de interesse, bem como mudanças drásticas de luminosidade que, muitas das vezes, induzem o modelo a criar uma previsão errada.

6.3.1 Filtros aplicados às *frames*

Perante esta situação, para obter melhores resultados nos modelos criados, devem ser aplicados diversos filtros de imagem como pré-processamento da mesma.

Em seguida, são indicados os diversos filtros utilizados e o seu objetivo, tendo em conta que são aplicados pela mesma ordem em que são apresentados:

Filtro	**Descrição**
BackgroudSubtractor MOG2	Método de extração de fundo estático, baseado na representação do movimento instantâneo entre duas *frames*. Isto acontece ao verificar se a variação do valor de cor entre os dois pixéis a ser processados. Se a mesma for superior a um *threshold* pré-definido, o pixel fica com valor 255 (branco), senão o pixel fica com o valor 0 (preto)
GaussianBlur	É utilizado para desfocar a imagem, removendo uma grande parte do ruído, através da aplicação de uma função *gaussiana*. Para a aplicação da função *gaussiana* é utilizado um *kernel* de 5x5, ou seja, um filtro de 5 pixéis de largura e de 5 pixéis de altura. Após a aplicação deste filtro, é diminuída a diferença entre cores semelhantes e aumentada a diferença entre cores distintas, com o intuito de melhorar os resultados da Rede Neuronal Convolucional a aplicar.
Threshold	Trata-se de um método para reforçar a diminuição da diferença entre cores semelhantes e o aumento da diferença entre cores distintas.
MedianBlur	Desfoca a imagem, mormente ao atribuir, a cada pixel, o valor médio do valor dos pixéis que o filtro (neste caso de 3x3) abrange.
morphologyEx	Remove pixéis isolados ao atribuir, aos mesmos, o valor mediano dos pixéis que o filtro

	abrange. Para este método, foram utilizadas duas iterações do mesmo, uma com um filtro de 4x4 e outra com um filtro de 3x3.

Tabela 14 - Filtros aplicados às imagens tratadas

Para a extração do fundo da imagem, foi, também, testada a criação de uma *frame* média ponderada, que seria subtraída à *frame* a ser processada, sendo que este método mostrou que seria ótimo para a vídeos de grande duração. No entanto, como o funcionamento ideal desta aplicação implica a recolha de vídeos de curta duração, foi decidida a utilização do método de subtração do fundo da imagem *Background Subtractor MOG2*, anteriormente descrito.

Após a aplicação de todos estes filtros, a imagem que obtida é, no fundo, uma representação instantânea da diferença entre duas *frames*.

Apresentam-se, em seguida, dois exemplos de imagens obtidas através da aplicação destes filtros:

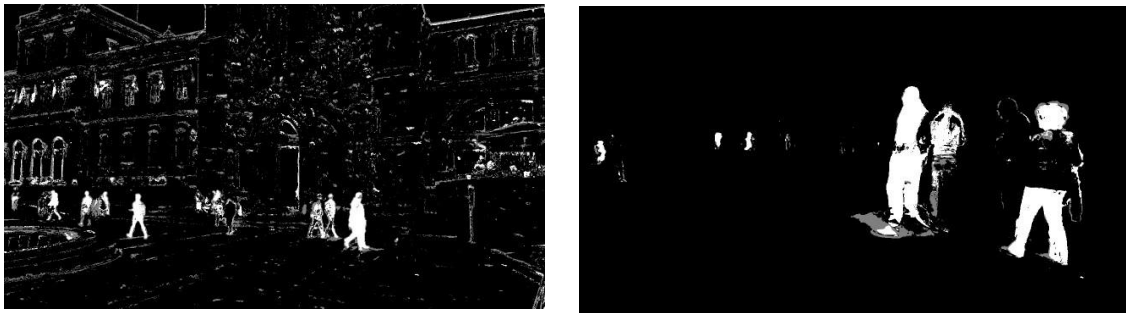


Figura 15 - Exemplos de imagens A (esquerda) e B (direita) do dataset criado

Tal como se pode verificar na figura 15a, é, por vezes, impossível eliminar todo o ruído, nomeadamente devido à enorme quantidade de cores diferentes, bem como às mudanças de cor, de luz ou de condições atmosféricas. No entanto, é possível diluir esse ruído e dar ênfase aos objetos que se pretende que sejam reconhecidos.

Na figura 15b, podemos ver que, se não existirem grandes quantidades de ruído, os filtros aplicados devolvem uma imagem em que a quantidade de pessoas é facilmente identificável, mas em que as características individuais das mesmas não o são.

6.3.2 Criação do *dataset*

Tendo em conta que os dados a utilizar neste projeto são sensíveis, o acesso aos mesmos foi muito difícil de conseguir.

Após diversos contactos com entidades que pudessem ceder este tipo de dados, procedeu-se ao contacto com o Doutor Humberto Rendeiro, Técnico Superior no Museu Monográfico e nas Ruínas de Conimbriga, que permitiu o acesso às Ruínas num dia 11 de Maio de 2018, nesse dia, deu-se a visita de diversos grupos de estudantes ao local, e durante essa mesma visita foi permitido, ao autor deste relatório, gravar vídeos de curta duração para a criação do *dataset*. Esses vídeos foram processados e a partir deles foram retiradas as imagens para o *dataset* (imagens apresentadas na figura 15) sendo, de seguida (e no respeito pela Política de Proteção de Dados), eliminados os vídeos originais, nos quais era possível distinguir características individuais das pessoas nele presentes.

Para um teste inicial da criação e implementação dos modelos, foi definida, após discussão com os membros da iClio, a utilização de cinco classes possíveis para a ocupação de um ponto de interesse, sendo as mesmas "*low_density*", "*medium_low_density*", "*medium_density*", "*medium_high_density*" e "*crowded*".

Para a criação das imagens através dos vídeos obtidos nas ruínas, foi necessário proceder-se à criação de uma diretoria na base do projeto designada de "*newDataset*", na qual foram criadas as subdiretorias "*low*", "*medium_low*", "*medium*", "*medium_high*" e "*crowded*", que correspondem às possíveis classes para os dados anteriormente apresentados, descritas na tabela seguinte:

Classe	**Descrição**
Low_density	Indica que um ponto de interesse pode receber novos visitantes sem qualquer tipo de restrições
Medium_low_density	Indica que um ponto de interesse ainda pode receber novos visitantes mas que está próximo de atingir o valor ideal de ocupação
Medium_density	Indica que um ponto de interesse está no nível ideal de ocupação
Medium_high_density	Indica que um ponto de interesse tem um nível de ocupação acima do ideal
Crowded	Indica que um ponto de interesse está completamente sobreocupado e que não devem ser aconselhados utilizadores a visitar o mesmo

Tabela 15 - Classes de densidade definidas para a Rede Neuronal Convolucional

Após esta estrutura de diretorias estar criada e as classes possíveis estarem definidas, procedeu-se ao tratamento e classificação manual dos vídeos obtidos nas ruínas, distribuindo-os pelas subdiretorias da classe associadas ao mesmo.

Foram, assim, reunidos todos os requisitos para se desenvolver um *script* de criação do *dataset* através dos vídeos tratados e classificados anteriormente, que lê todos os vídeos para cada classe, aplica os filtros previamente mencionados e exporta as *frames* para uma diretoria "*dataset*", já com os dados prontos a realizar um treino do modelo.

O código associado a este *script* de criação de *datasets* pode ser encontrado no anexo 21.

Para uma maior definição e precisão no modelo final, assume-se que as classes devem ter um número de amostras semelhante. Neste pressuposto, e tendo em conta que os vídeos obtidos não tinham o mesmo número de *frames* para cada classe, foi necessário criar uma fórmula para a distribuição das amostras.

A fórmula utilizada foi a seguinte:

$$X = MED / nº \text{ de frames da classe}$$

Assim, pode-se equilibrar o número de amostras das classes com o valor de:

Número de Frames em que se exporta 1 frame final = i / X

Em que i = 10 significa que deve ser exportada 1 em cada 10 *frames*.

Para uma melhor compreensão da fórmula utilizada, apresenta-se, a seguir, um exemplo.

Consideremos que as 5 classes têm um número de *frames* total em todos os vídeos associado de:

Classe	Número de frames
A	500
B	400
C	200
D	50
E	50

Tabela 16 - Número total de frames para cada classe no exemplo a ser descrito

Consideremos, também, que o valor de i para este caso é de 10, ou seja, que deve ser exportada uma *frame* em cada 10 em classes com número de amostras igual.

Sabemos assim que o número médio de *frames* é:

$$MED = (500+400+200+50+50) / 5 = 200 \textit{ frames}$$

Assim, é-nos, já, possível calcular o valor de X da fórmula anteriormente mencionada:

$$X1 = 200/500 = 0.4$$

$$X2 = 200/400 = 0.5$$

$$X3 = 200/200 = 1$$

$$X4 = X5 = 200/50 = 4$$

Com os valores de X, podemos calcular o número de *frames* finais exportadas para cada classe, que se apresenta semelhante em todas, tal como se pode verificar nos cálculos apresentados de seguida:

Classe	**Exporta 1 *frame* em cada:**	**Número de *frames* final**
A	10 / 0.4 = 25	500 / 25 = 20
B	10 / 0.5 = 20	400 / 20 = 20
C	10 / 1 = 10	200 / 10 = 20
D	10 / 4 = 2.5 (arredondado para 3)	50 / 3 = 16
E	10 / 4 = 2.5 (arredondado para 3)	50 / 3 = 16

Tabela 17 - Número de frames final para cada classe no exemplo a ser descrito

Assim, obtém-se, por fim, 1 diretoria designada de *dataset*, constituída por 5 subdiretorias, referentes às 5 classes referentes à densidade populacional de um ponto de interesse, sendo as mesmas designadas de *LOW0*, para baixa densidade populacional, *MEDIUMLOW1*, para média baixa densidade populacional, *MEDIUM2*, para a média densidade populacional, *MEDIUMHIGH3*, para a média alta densidade populacional, e *CROWDED4*, classificação atribuída a pontos de interesse que já não tenham capacidade para receber mais turistas, todas com um número semelhante de amostras.

6.3.3 Treino do modelo

Com o *dataset* criado, pode proceder-se ao treino do modelo.

Para a realização deste passo, foi necessário criar dois módulos: o "*CNNmodeler.py*" e o "*Train.py*".

No módulo "*CNNmodeler.py*" têm que estar presentes a definição da Rede Neuronal Convolucional, as várias camadas que a mesma vai possuir, bem como todos os parâmetros associados, anteriormente mencionados.

Para o caso aqui descrito, a Rede Neuronal Convolucional vai apresentar as seguintes características:

- Entrada de imagens com dimensão [360,640,3], ou seja, 360 pixéis de altura por 640 pixéis de largura, e com 3 dimensões de valor de cor (*Red*, *Green* e *Blue*);
- Camada de Convolução com um filtro de 8x8, *padding* de 3 e *stride* de 1 e ativação *ReLu*;
- Camada de *Max Pooling* com um filtro de 2x2 e 2 de *stride*;
- Camada de Convolução com um filtro de 8x8, *padding* de 3, *stride* de 1 e ativação *ReLu*;
- Camada de *Max Pooling* com um filtro de 2x2 e 2 de *stride*;
- Camada de *Fully Connected* com 1024 neurónios e ativação *TanH*;
- Camada de *dropout* de 50% dos neurónios;
- Camada de *Fully Connected* com 10 neurónios (um por cada classe existente no *dataset*) e ativação *softmax*.

O código deste módulo pode ser visualizado no anexo 22.

Com a Rede Neuronal Convolucional definida, pode, então, proceder-se à criação do módulo de treino do modelo, designado de "*train.py*", para o qual foi necessário:

- Importar o módulo da Rede Neuronal Convolucional;
- Criar uma função para eliminar o ficheiro "*DS_Store*" (ficheiro de sistema do *MacOS*) de todas as diretorias e subdiretorias do *dataset*;
- Criar um ficheiro *HDF5* para o *dataset*, processo que pode ser visualizado na seguinte figura:

```
# Build a HDF5 dataset (only required once)
from tflearn.data_utils import build_hdf5_image_dataset
build_hdf5_image_dataset(root_path, image_shape=(640, 360), grayscale=True, mode='folder', output_path='dataset.h5', categorical_labels=True, normalize=True)
print('HDF5 file created for Training Dataset')
```

Figura 16 - Criação do dataset em HDF5

- Ler o ficheiro *HDF5* criado no passo anterior e projetar as 3 dimensões de cor para uma única, com recurso à função *reshape* da biblioteca *numpy* do *python*, como se pode verificar na seguinte figura:

```
#Read HDF5 dataset
import h5py
h5f = h5py.File('dataset.h5', 'r')
(X,Y) = (h5f['X'],h5f['Y'])
X = np.reshape(X, (-1, 360, 640,1))
print('Training dataset read from HDF5!')
Y.shape #shape the numpy array
print ('data processing complete!\nStarting to train model, this may take a while...')
```

Figura 17 - Leitura do ficheiro HDF5

- Aplicar a Rede Neuronal Convolucional definida no módulo "*CNNmodeler.py*" ao *dataset* criado e guardar os modelos resultantes. Para o modelo resultante ser o mais preciso possível, foram definidas 15 iterações de treino, 30% dos dados do *dataset* foram definidos como dados para validação do modelo (como parte do *Validation Set*) e foi definida, como métrica de comparação entre modelos, a Precisão de Classificação, tal como se pode verificar na seguinte figura:

```
model.fit(X,Y, n_epoch=15, validation_set=0.3, show_metric=True, run_id="CNN_Modeler")#15 loops, 30% of data as validation set and metric Accuracy
model.save('final-model.tflearn')#exports the final model with a different name
```

Figura 18 - Treino da Rede Neuronal Convolucional

Com o módulo de treino funcional, pode-se proceder à execução dos módulos criados ao inserir no terminal, dentro do ambiente virtual criado previamente e dentro da diretoria em que se encontram os módulos criados, o seguinte comando:

python train.py

Dentro do terminal, podemos visualizar o progresso do treino do modelo e obter diversas informações do mesmo. As medidas de precisão de classificação e o tempo de execução do treino do modelo apresentados anteriormente podem ser visualizados no anexo 23.

Para o caso de estudo a ser descrito, o *dataset* final é constituído por:

- 1303 amostras para o treino do modelo;
- 559 amostras para o teste do modelo.

Tal como mencionado anteriormente, o treino de modelos baseados em Redes Neuronais Convolucionais é um processo demorado e que consome muitos recursos da máquina em que está a ser executado.

No caso de estudo aqui descrito, foram obtidos os seguintes resultados:

- Tempo de execução:

14268.740 + 13073.059 + 14076.961 + 13154.475 + 14235.503 + 15204.317 + 14096.677 + 14016.568 + 14514.666 + 14733.570 + 13951.030 + 14444.682 + 14410.371 + 14427.140 + 14693.741 =

= 213301.5 s = 3555 minutos e 1.5 segundos = 59:15:1,5 h (2 dias, 11 horas, 15 minutos e 1.5 segundos)

- Iteração com maior valor de precisão e menor valor de perda de informação:

Iteração número 13: 0.9624 de precisão e 0.13124 de perda de informação na classificação do *dataset* de validação

Como observação importante, devemos referir que, tendo em conta que a máquina que está a executar o treino do modelo não pode ser utilizada para outros propósitos, sob o risco de o treino falhar, nomeadamente por falta de recursos disponíveis, de forma temporária, por parte da máquina, e que o tempo de execução era superior a dois dias, mostrou-se impossível o treino de diversos modelos para a obtenção dos parâmetros ideias para o caso de estudo, como seria desejável, sendo que o treino de modelo fica, assim, no formato de prova de conceito, em que está completamente funcional. Porém, isso não implica que não requeira o teste de diferentes valores dos diferentes parâmetros da Rede Neuronal Convolucional, para tentar obter o classificador que apresente a melhor precisão de classificação e a menor perda de informação possível.

6.4 Funcionamento do modelo

Para um melhor funcionamento do modelo e para garantir o cumprimento da *RGPD*, foram criadas duas diretorias para ficheiros temporários dentro da diretoria raiz do projeto, sendo elas a diretoria *tempVideos* e a diretoria *tempImages*.

Assim, o funcionamento ideal da aplicação será o apresentado na seguinte figura:

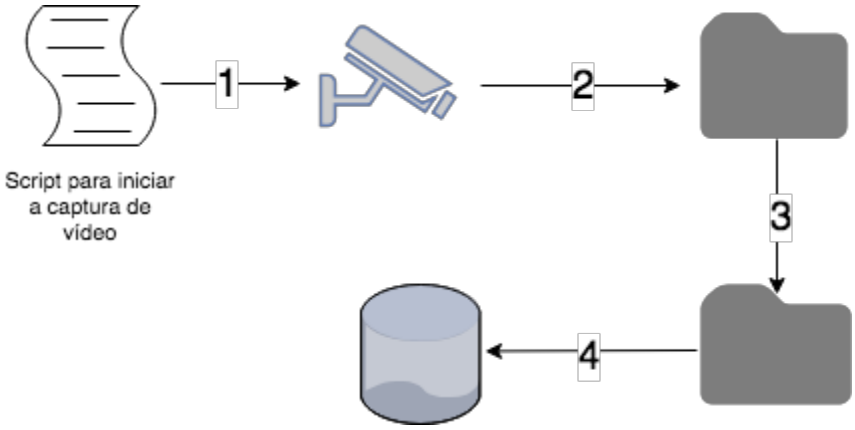


Figura 19 - Esquema de funcionamento dos modelos criados

Esta figura pode ser explicada através dos seguintes pressupostos:

1. Dentro de um *CronJob* (um método de correr *scripts*, automaticamente, a cada período de tempo previamente definido, como, por exemplo, a cada hora), deve ser executado o *script* de captura de vídeo para cada câmara;
2. Para cada câmara, é necessário guardar o vídeo associado à mesma na diretoria "*tempVideos*", sendo que o nome do vídeo deve incluir o endereço *IP* da câmara a que pertence, seguindo o formato "1.1.1.1.*mp4*" e que o mesmo deve conter, pelo menos, 200 *frames* (3.33 segundos de vídeo a 60 *frames* por segundo);
3. Para cada um dos vídeos, tem que se aplicar o *script* de criação de imagens (cujo código se apresenta no anexo 24) para o mesmo. Este *script* é constituído por 2 principais módulos: o módulo de aplicação dos filtros descritos anteriormente e o módulo de validação do vídeo obtido, que é um módulo que tem como objetivo controlar a qualidade dos vídeos que devem ser classificados. Para uma explicação mais simplista deste módulo, podemos referir que, se a diferença entre a área máxima e mínima branca das imagens resultantes da aplicação dos filtros a todas as *frames* do vídeo (ou seja, área em que existe movimento) for inferior a 25%, o vídeo é validado; se o vídeo não for validado, é recolhido um novo vídeo para essa câmara, processo que se repetirá tantas vezes quantas as necessárias até haver um que seja validado. Esta validação tem como objetivo remover os dados que posssam induzir o classificador em erro, isto é, dados que provenham, por exemplo, de grandes discrepâncias de luminosidade ou um grupo de pessoas a passar por um ponto de interesse sem parar no mesmo. As imagens resultantes dos vídeos validados são guardadas na diretoria "*tempImages*" com o nome "1.1.1.1_número da imagem.jpg".

4. Por fim, deve aplicar-se o modelo criado anteriormente às imagens presentes na diretoria "*tempImages*". Numa tentativa de melhorar a precisão da classificação final das imagens, é aplicado o modelo ao grupo de imagens obtido para cada uma das câmaras, sendo o resultado final a moda de classificações obtidas para essa câmara. Após a classificação estar completa, é criado um pedido ao servidor para a rota de adicionar a previsões apresentada anteriormente. O código do módulo "*Predictor.py*" pode ser visualizado no anexo 25.

Com todos estes módulos criados e com o funcionamento proposto implementado, a nossa aplicação é capaz de classificar a densidade populacional de um ponto de interesse, podendo esta informação ser utilizada pelos algoritmos de rotas propostos em seguida.

6.5 Algoritmos de rotas

A funcionalidade essencial da aplicação desenvolvida implica a criação de rotas turísticas para o utilizador, rotas essas que podem ser personalizadas ou automáticas, baseadas no tempo e na posição inicial do mesmo.

Apesar destes algoritmos apenas estarem implementados na *Frontend*, que descreveremos no capítulo seguinte, o estudo, a planificação e o funcionamento dos mesmos são problemas de otimização, sendo o mais conhecido o *TSP* (*Travelling Salesman Problem*), para o qual existem inúmeras propostas de algoritmos. De entre as diferentes possibilidades existentes, foram retiradas informações sobre os algoritmos implementados, bem como sobre os algoritmos que foram estudados e não foram implementados, pelo que consideramos que se enquadram neste capítulo.

Com efeito, os algoritmos criados no projeto aqui descrito foram baseados nos algoritmos *Nearest Neighborgs*. De acordo com este pressuposto, consideramos um algoritmo que, após ser escolhido um ponto inicial, cria uma rota em que o ponto seguinte é sempre o ponto mais próximo, bem como a Pesquisa Exaustiva, que pesquisa, entre todas as rotas possíveis, qual a implica menor distância, retornando a mesma.

O algoritmo de Pesquisa Exaustiva baseia-se na utilização de uma lista que contém todas as permutações possíveis dos pontos de interesse presentes na "*openList*". Esta lista será preenchida com os identificadores únicos (*ID*s) dos pontos de interesse e terá o seguinte aspeto:

1	**2**	**3**	**4**	**...**	**k**
1	**2**	**4**	**3**	**...**	**...**
1	**3**	**2**	**4**	**…**	**...**
...	**…**	**...**	**...**	**...**	**...**
k	**1**	**2**	**3**	**...**	**k-1**

Tabela 18 - Exemplo de uma "openList" com todas as permutações de 1 até k

Com esta lista, é possível implementar o algoritmo de Pesquisa Exaustiva:

- ao criar um ciclo que itere por cada uma das colunas;
- ao calcular a distância total de cada uma das mesmas;
- ao retornar a rota com o menor valor de distância total.

Apesar de estes algoritmos serem bastante conhecidos e utilizados, ambos apresentam pontos fortes (vantagens) e pontos fracos (desvantagens) no seu uso. Na tabela seguinte podemos ver, de forma sintetizada, essas mesmas vantagens e desvantagens:

Algoritmo	Vantagens	Desvantagens
Nearest Neighbor	É rápido.	Escolhe o ponto seguinte de forma muito básica (apenas baseada na distância), tendo a tendência a seguir ótimos locais (rotas em que a combinação dos primeiros pontos é ótima, mas em que a combinação total dos pontos não o é).
Pesquisa Exaustiva	Garante o resultado ótimo.	É lento, mormente devido ao crescimento exponencial de combinações quando estabelece a rota consoante o número de pontos de interesse da mesma.

Tabela 19 - Vantagens e desvantagens dos algoritmos de rotas aplicados

Tendo em conta que a base de dados, o servidor e os modelos de classificação de taxa de ocupação estão, neste ponto, funcionais e em execução, pode proceder-se à planificação do funcionamento geral dos algoritmos de rotas, sendo que, baseado nos requisitos de as rotas poderem ser personalizadas ou automáticas, baseadas no tempo e na posição inicial do utilizador, foram decididos dois tipos de funcionamento principais, que a seguir apresentamos.

6.5.1 Algoritmo de rotas personalizado

No algoritmo de criação de rotas personalizado, no âmbito do qual o utilizador define a cidade e as categorias de pontos de interesse que pretende visitar. Paralelamente, o mesmo utilizador seleciona um ponto de interesse como ponto inicial da rota e outro como ponto final, sendo, assim, possível criar uma rota que minimize a distância percorrida, que inclua todos os pontos de interesse selecionados pelo utilizador, que tenha em conta a taxa de ocupação registada pelas câmaras e o tempo médio de visita de cada ponto de interesse e que devolva a distância e a duração total da rota.

Tendo em conta que as rotas irão ter que ter em consideração a taxa de ocupação de um ponto de interesse registada pela câmara associada ao mesmo, foi decidida a introdução do conceito das listas de pontos a explorar, designada de "*openList*", e de pontos à espera que a taxa de ocupação dos mesmos diminua, designada de "*waitList*".

Tal como os nomes indicam, a "*openList*" é a lista da qual pode ser escolhido o próximo ponto da rota que está a ser criada e a "*waitList*" é a lista onde estão os pontos de interesse cujo valor de taxa de ocupação mais recente registado na base de dados seja "*high density*" ou "*crowded*". Um ponto é removido da "*waitList*" e adicionado à "*openList*" assim que o tempo acumulado da rota criada seja superior ao seu tempo médio de visita, pois assume-se que, se os utilizadores não são aconselhados a visitar um ponto de interesse, o mesmo irá descer a sua taxa de ocupação para níveis aceitáveis (abaixo de "*high_density*").

Tendo em conta a introdução deste conceito, foi decidida a utilização de ambos os algoritmos – *Nearest Neighbor* e Pesquisa Exaustiva –, na tentativa de que se complementem, privilegiando e, consequentemente, utilizando as vantagens e diminuindo as desvantagens de cada um deles.

Para isso, foi decido que o funcionamento do algoritmo para a aplicação desenvolvida seria a utilização do algoritmo *Nearest Neighbor* enquanto a "*waitList*" não estiver vazia sendo que, se isto acontecer, será utilizado o algoritmo de Pesquisa Exaustiva.

Este funcionamento está representado na seguinte figura:

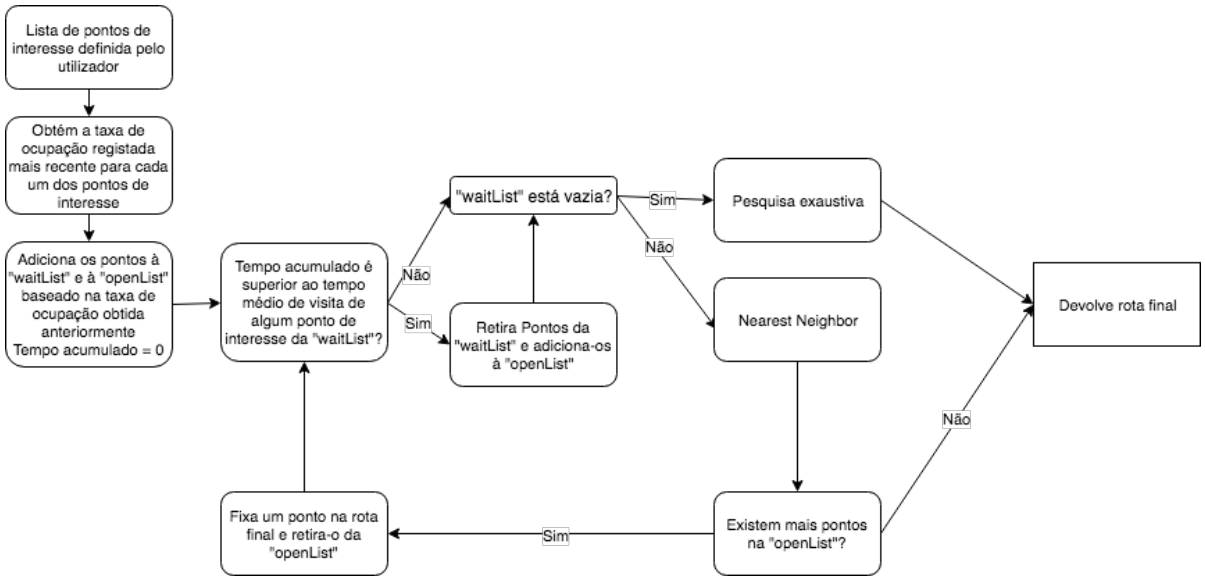


Figura 20 - Esquema de funcionamento do modo personalizado de criação de rotas

6.5.2 Algoritmo de rotas automático

Trata-se de um algoritmo de criação de rotas automático em que o utilizador define a cidade que pretende visitar, o tempo que tem disponível para a sua rota turística e a sua posição inicial, de forma a que seja criada uma rota que inclua o máximo de pontos de interesse possível, que tenha em conta a taxa de ocupação registada pelas câmaras, o tempo médio de visita de um ponto de interesse e a sua prioridade de visita.

Para tal, depois do utilizador escolher este modo e definir a duração da rota que deseja criar, a aplicação irá:

1. obter a localização do utilizador;
2. obter o ponto de interesse com menor distância à posição do utilizador e com prioridade superior a 4 (escala de 1-5);
3. dividir os pontos de interesse pelas listas "*openList*" e "*waitList*" (tal como no algoritmo personalizado);
4. obter uma lista com todos os pontos de interesse de prioridade 4 e 5 presentes na "*openList*";
5. obter todas as permutações da lista obtida em (4);
6. para cada uma das permutações, terá as seguintes funções:

- se ponto inicial for o ponto obtido em (2), vai calcular o número de pontos da permutação cuja duração acumulada da rota seja inferior a dois terços da duração definida pelo utilizador;
- regista o máximo histórico de pontos que pode visitar e a rota associada à mesma; se houver vários registos com o mesmo número de pontos possível, será registado o valor mínimo de distância e a rota associada ao mesmo.

7. para cada um dos pontos de prioridade baixa (1 a 3), guardar qual o ponto da permutação ideal obtida em (6) mais próximo;
8. verificar qual o custo de adicionar cada ponto obtido em (7) na rota obtida em (6) antes e depois da sua posição na rota, guardando o valor mínimo. Por exemplo, se o ponto mais próximo do ponto de interesse A for o ponto X, será calculado o custo de adicionar o ponto X antes do ponto A, apresentando a rota X – A - … ; outra hipótese possível será, depois do ponto A, apresentar a rota A – X - …. ; em qualquer dos casos, será guardado o valor mínimo destes dois custos calculados.
9. para todos os resultados obtidos em 8, adicionar os mesmos à rota final, ordenados por custo ascendente, até que a duração da mesma seja igual à duração definida pelo utilizador.

O esquema de funcionamento deste algoritmo é a que a seguir se apresenta:

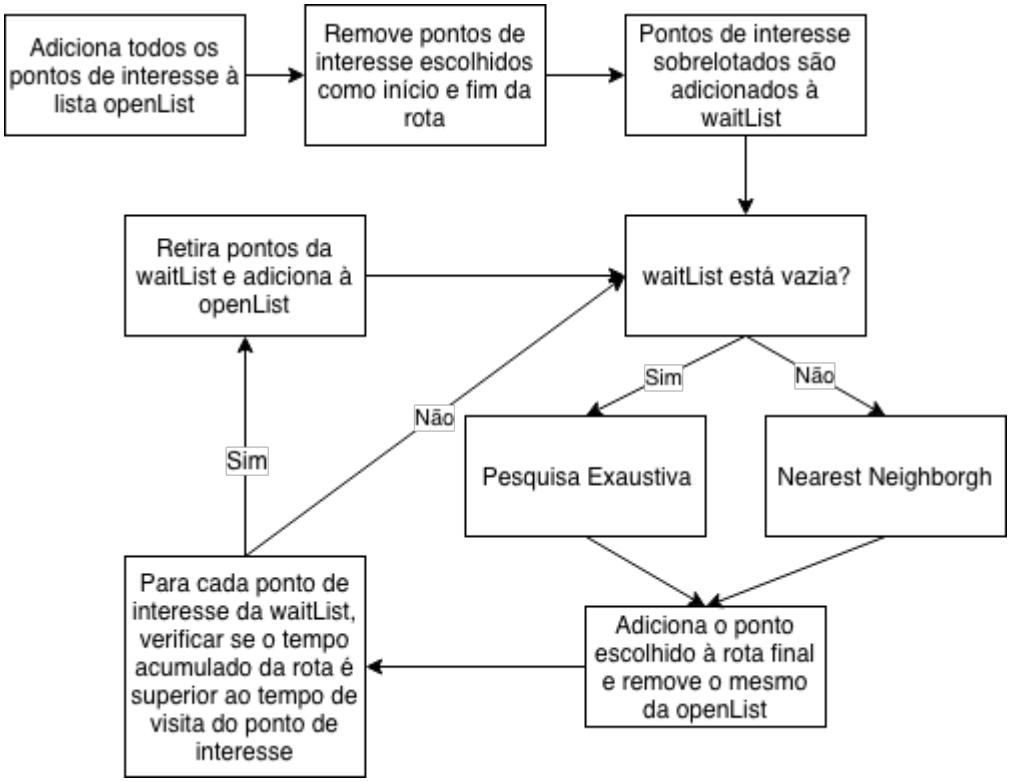


Figura 21 - Funcionamento do algoritmo de rotas personalizado

Com ambos os algoritmos planificados e o seu funcionamento claramente definido, e tendo em consideração todos os módulos apresentados anteriormente e em execução, é possível proceder à criação da *Frontend*, o último módulo da aplicação a desenvolver e que requer o funcionamento de todos os outros módulos para operar.

7 Frontend

A *Frontend* será, na aplicação aqui descrita, a implementação em *vueJS* e *nodeJS* de uma interface gráfica que permita tirar partido de todos os módulos anteriormente descritos, nomeadamente a *API RESTful*, os modelos de classificação de taxa de ocupação de um ponto de interesse e os algoritmos de rota.

Relativamente ao *VueJS*, trata-se de uma biblioteca que permite a criação de uma aplicação *web* de página única, mas que permite a navegação e a mudança de *templates*, através de componentes individuais que podem ser utilizadas na página principal, tanto sozinhas como em grupo, de forma a controlar o conteúdo e as funcionalidades da página que é mostrada ao utilizador durante a sua navegação pela aplicação.

Devido ao funcionamento do *VueJS*, foi necessário, numa primeira fase, proceder à definição dos requisitos de funcionalidades gerais da aplicação *Web*, de forma a decidir as componentes necessárias à sua implementação. Os requisitos de funcionalidades e as componentes utilizadas, bem como qual o tipo de utilizador que tem acesso aos mesmos são os que a seguir se apresentam, de forma simplificada e devidamente esquematizada, para mais fácil visualização e compreensão:

Funcionalidade	**Descrição**	**Componentes**	**Utilizador Normal**	**Admin.**
Aplicação *Web*	Página base, que irá conter todas as componentes criadas	*App* – é a componente "pai" de toda a aplicação *Web*; nela consta o menu de navegação (comum a todas as outras componentes) e irá utilizar todas as componentes a seguir definidas.	X	X
Autenticação com *tokens* de sessão	A aplicação *Web* deverá permitir o registo e autenticação de utilizadores, sendo que também é implemento o conceito de *token* de sessão, cujo funcionamento já foi explicado no capítulo referente à base de dados	*Register* – Componente de registo de um novo utilizador; *Login* – Componente de autenticação de um utilizador.	X	X

Homepage	Página inicial que é apresentada na aplicação	*Home* – Primeira componente apresentada ao utilizador na aplicação *Web*, implementada, mas sem conteúdo.	X	X
About	Página com informação sobre os contactos	*About* – Componente de apresentação da informação da Entidade que mantém a aplicação *web*.	X	
NotFound	Página retornada sempre que o utilizador tenta navegar para um componente que não exista	*NotFound* – Componente que apenas apresenta, ao utilizador, um erro "404 *Not Found*", caso se trate de uma componente inexistente.	X	X
Visualizar, editar e eliminar informação da base de dados	Diversas páginas a mostrar ao utilizador, de forma a permitir a interação com a base de dados	*POI*, *Camera*, *Category* e *City* – Componentes de visualização, edição e remoção de pontos de interesse, câmaras, categorias e cidades, respetivamente, na base de dados; *AddPOI*, *AddCamera*, *AddCategory* e *AddCity* – Componentes de inserção de um novo registo de um ponto de interesse, uma câmara, uma categoria e uma cidade, respetivamente, na base de dados; *Prediction* – Componente que apenas permite a visualização dos dados de previsões associadas às câmaras registadas na base de dados; a inserção de dados é feita, automaticamente, através do funcionamento do classificador e a edição e remoção de dados é impossível; *User* – Componente que permite a visualização de utilizadores e a edição do seu "*role*" na aplicação *Web*. A inserção de novos utilizadores é feita através do registo dos mesmos.		X
Route	Página que permitirá, ao utilizador, criar a sua própria rota (personalizada ou	*Route* – Componente que contém um menu de navegação lateral e um mapa, que deverá permitir a apresentação, ao utilizador, dos		

	automática, sempre baseada no tempo), onde serão implementados os algoritmos de rotas anteriormente descritos	pontos de interesse e da rota criada.	X	

Tabela 20 - Requisitos de funcionalidades e componentes para a aplicação desenvolvida

Com esta estrutura de componentes definida e que permite corresponder a todos os requisitos funcionais da aplicação *Web*, foi possível proceder à implementação dos mesmos.

7.1 Criação do projeto da *Frontend*

A *frontend*, tal como descrito anteriormente, está assente na tecnologia *VueJS*, que permite a criação de aplicações *web* vazias, com todos os ficheiros necessários ao funcionamento das mesmas, através do processo descrito em seguida:

1. Abrir o terminal e navegar para a subdiretoria *Frontend* criada anteriormente dentro da diretoria base do projeto;
2. Inserir o seguinte comando no terminal:

vue init webpack "Nome do Projeto".

Para o projeto aqui descrito, foi utilizada a funcionalidade de inicialização de um *webpack*, isto é, de um módulo estático para aplicações *JavaScript* que processa a aplicação criada, cria um gráfico de dependências que mapeia cada módulo que é utilizado na aplicação a ser desenvolvida e cria um ou mais *bundles* que as agrupe(m) a todas.

Com a aplicação base criada e pronta a ser utilizada, devem ser importadas as diversas bibliotecas que irão ser utilizadas pela mesma com recurso ao *NPM* (*Node Package Manager*), uma funcionalidade do terminal que permite a instalação de bibliotecas externas na aplicação a desenvolver, mantendo um registo de todas as bibliotecas e dependências da aplicação no ficheiro "*package.json*" que, a cada execução da aplicação, é compilado, para garantir que a mesma tem acesso a todos os módulos necessários.

Deve, então, proceder-se à inserção do comando "*npm run dev*" dentro da diretoria do projeto *Frontend* que contenha o ficheiro "*package.json*", para que a aplicação seja executada e possa ser acedida através do URL http://localhost:8080/.

Após a instalação das diversas bibliotecas mencionadas no capítulo relativo às tecnologias utilizadas, foi possível proceder à criação das componentes necessárias ao funcionamento da aplicação, que a seguir se apresentam, de forma individualizada.

7.2 Página principal

Tal como foi mencionado anteriormente, o *VueJS* baseia-se numa página principal, que será sempre apresentada e que irá incluir as componentes que deverão ser apresentadas ao utilizador, aquando a sua navegação pela aplicação.

Esta página será apenas constituída pelo menu de navegação da aplicação. Para cumprir os requisitos de autenticação e sessão de utilizadores, bem como de restrição de acesso dos utilizadores às funcionalidades da aplicação consoante o seu "*role*", foi decidida a utilização da seguinte estrutura:

- Um botão de navegação para a "*Home*";
- Um botão para interagir com os dados da base de dados, disponível apenas para utilizadores com o "*role*" de '*admin*', que fará aparecer uma lista das diversas tabelas criadas e permitir a navegação para as respetivas componentes;
- Um botão para navegar para a componente na qual estão implementados o mapa e os algoritmos de criação de rotas, disponível apenas para os utilizadores normais;
- No caso de não haver uma sessão ativa:
    - Um botão para navegar para a componente de registo de um novo utilizador;
    - Um botão para navegar para a componente de login.
- No caso de haver uma sessão ativa:
    - Uma *label* que apresente o *email* do utilizador que tem a sessão ativa;
    - Um botão que termine a sessão do utilizador.

Para a implementação deste menu, visualizar o código apresentado no anexo 26.

7.3 Router

Com a componente "pai" definida e criada, e tendo em conta que esta irá conter as componentes que foram definidas na tabela 20, é necessário proceder à definição do roteamento das componentes, ou seja, a definição das palavras-chave que irão ser colocadas no *URL* para

navegar entre componentes, de forma a poder ocultar ou mostrar funcionalidades ao utilizador, consoante a componente a que o mesmo deseja aceder.

Para tal, foi necessário alterar o ficheiro presente dentro da subdiretoria "*router*", criada automaticamente pelo *webpack*, bem como implementar as seguintes rotas:

Rota da aplicação	**Componente**
'/'	Home
'/about'	About
'/register'	Register
'/login'	Login
'/poi'	POI
'/poi/add'	AddPOI
'/camera'	Camera
'/camera/add'	AddCamera
'/prediction'	Prediction
'/category'	Category
'/category/add'	AddCategory
'/city'	City
'/city/add'	AddCity
'/users'	Users
'/route'	Route
'*'	NotFound

Tabela 21 - Componentes criadas para a aplicação desenvolvida

O código de implementação do "*Router*" pode ser visualizado no anexo 27 .

7.4 Componente *Register*

Esta componente irá conter a funcionalidade de registo de novos utilizadores e estará acessível, sempre que não exista uma sessão de um utilizador ativa de momento, através do menu de navegação da componente "pai".

Esta página é constituída pelos seguintes elementos:

- Um campo para o utilizador inserir o seu *email*;
- Um campo para o utilizador inserir a *password* desejada;
- Um botão para finalizar o registo.

Na figura seguinte, é apresentado o aspeto final desta componente, na qual será implementada a funcionalidade de registo de um novo utilizador:

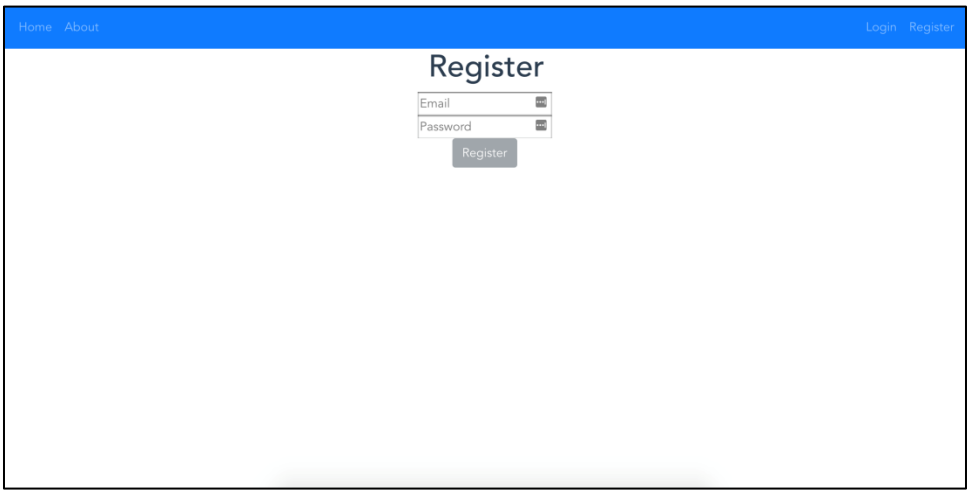


Figura 22 - Aspeto da componente Register

Tendo em conta que esta componente irá, no fundo, inserir informação relativa a um novo utilizador na base de dados, sabemos que será necessário criar dois pedidos para o servidor, nomeadamente para os caminhos "/register/uniqueEmail", que valida se o utilizador está a inserir um *email* que ainda não existe na base de dados, e "/register/complete" que, através do método *POST*, insere a informação do utilizador na base de dados.

É, no entanto, boa prática não criar uma ligação direta entre a *Frontend* e um pedido do tipo *POST* ou *PUT* à base de dados e recorrer à implementação de serviços para esta ligação que, apesar de não serem essenciais ou difíceis de implementar, impedem o acesso direto do

utilizador ao servidor (e o conhecimento das rotas do servidor pela parte do mesmo) e, consequentemente, à base de dados, no sentido de dificultar o mais possível os planos de utilizadores mal intencionados.

A implementação de serviços é apresentada de seguida, utilizando a inserção de um novo utilizador como exemplo:

```
import Api from '@/services/Api'

export default {
  addUserInfo (userInfo) {
    return Api().post('/register/complete', userInfo)
  }
}
```

Figura 23 - Serviço de registo implementado

Tal como se pode verificar, o objeto *userInfo*, que contém o *email* e a *password* que o utilizador inseriu previamente, será enviado para a rota do servidor '/register/complete', o que irá introduzir o novo utilizador na base de dados.

Com este serviço implementado, podemos utilizar o mesmo na *frontend*, ao importar o mesmo e construir o objeto *userInfo* anteriormente mencionado.

Qualquer tipo de interação que requeira o envio de informação inserida pelo o utilizador para o servidor deve apresentar um serviço de implementação do mesmo, sendo que, para esta componente, também seria necessário a criação de um serviço que verificasse se o *email* não está associado a um utilizador existente na base de dados, tal como mencionado anteriormente, devido à necessidade de enviar o *email* inserido pelo utilizador no corpo do pedido. O código associado a este serviço pode ser encontrado no anexo 28 .

Destacam-se, também, a criação de algumas funções de suporte ao funcionamento do registo, sendo as mesmas:

- Função ValidateEmail() - verifica se um *email* é válido ou não;
- Função ValidatePassword() - verifica se uma *password* tem um tamanho mínimo de 5 e um máximo de 15 caracteres;
- Função submitValidated() - verifica se o *email* e a *password* são ambos válidos. Se sim, permite a utilização do botão de finalização de registo;

- Função verifyUniqueEmail() - utiliza o serviço de verificação de se o *email* inserido no registo de um novo utilizador não está associado a um utilizador, e retorna a informação como verdadeira ou falsa;
- Função RegisterComplete() - se o serviço de verificação do *email* inserido no registo de um novo utilizador não está associado a um utilizador e retornar o valor "verdadeiro", então o novo utilizador é inserido na base de dados.

O código associado a este componente pode ser encontrado no anexo 29.

7.5 Componente *Login*

Em termos de elementos presentes nesta componente, ela é idêntica à componente de registo, na medida em que também possui apenas um campo para o utilizador inserir o seu *email*, um campo para o utilizador inserir a sua *password* e um botão para efetuar a autenticação na aplicação *Web* desenvolvida.

No entanto, os serviços e funções utilizadas pela componente *Login* diferem da componente de Registo, na medida em que é implementada uma única função com um serviço associado para a autenticação do utilizador, designado de *LoginService*, que funciona como intermédio entre a componente a descrever e o caminho do servidor '/login/complete' e cujo código pode ser encontrado no anexo 30.

Tendo a componente, a função de autenticação da mesma e o serviço de autenticação implementados, a página de *login* resultante é a que agora se apresenta:

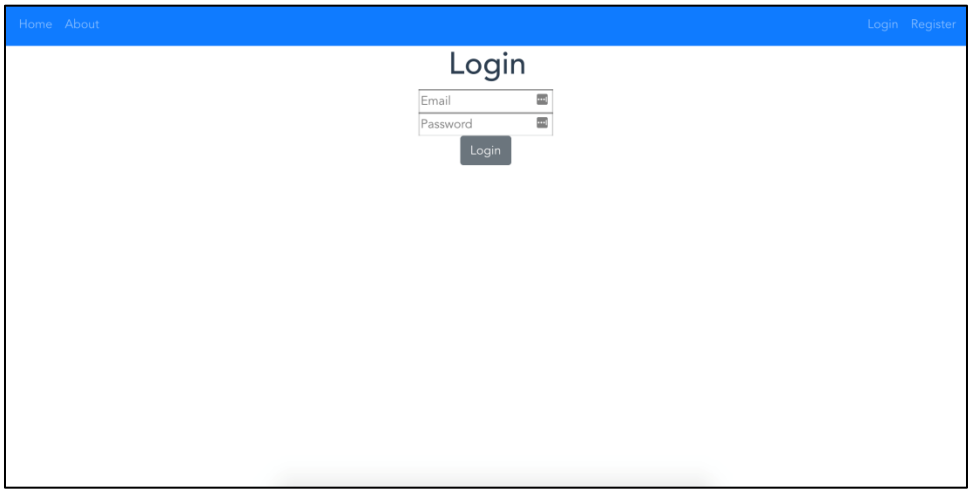


Figura 24 - Aspeto da componente de Login

O código associado a esta componente pode ser encontrado no anexo 31.

7.6 Componente *POI*

A componente *POI* será a componente principal de interação com a tabela *POI* da base de dados. Esta deverá permitir a visualização de todos os pontos de interesse, a pesquisa pontos de interesse através do seu nome, a edição e remoção de pontos de interesse de bases de dados, bem como conter um botão de acesso à componente *AddPOI*, anteriormente descrita. Para cumprir todos estes requisitos, esta componente será constituída pelos seguintes elementos:

- uma barra de pesquisa que permita a pesquisa de pontos de interesse pelo seu nome;
- um botão que permita a navegação para a componente *AddPOI*;
- uma tabela com que apresente a informação de cada um dos pontos de interesse e que, nas duas últimas colunas da tabela, contenha os botões para edição e remoção de um ponto de interesse;
- no caso de se selecionar a funcionalidade de edição de um ponto de interesse, os elementos da componente inicialmente apresentados ao utilizador devem ser escondidos e deverão ser apresentados diversos elementos para a inserção dos diversos atributos do ponto de interesse, que devem ser preenchidos automaticamente com a informação existente do ponto de interesse, para que possam ser alterados pelo utilizador, consoante o seu agrado;
- no caso de se selecionar a funcionalidade de remoção de um ponto de interesse, deve ser apresentado um aviso em que o utilizador deve confirmar ou cancelar a remoção desse mesmo ponto de interesse.

Após os elementos a utilizar estarem definidos, foi necessária a implementação dos serviços:

Serviço	**Descrição**	**Anexo em que está apresentado**
FindByNameService	Serviço que permite a pesquisa de um ponto de interesse através do seu nome, enviando o conteúdo do elemento barra de pesquisa, anteriormente mencionado no corpo do pedido, ao caminho do servidor '/pois/findbyname'.	32
UpdatePOIService	Serviço que permite a alteração dos dados de um ponto de interesse e que deverá esconder todos os elementos mostrados inicialmente ao utilizador, bem como mostrar os elementos de	

	inserção de dados relativos a um ponto de interesse, com recurso ao caminho do servidor '/pois/update'.	33
DeletePOIService	Serviço que permite a remoção dos dados do ponto de interesse cujo *ID* é enviado como parâmetro do pedido, com recurso ao caminho do servidor '/pois/delete'.	34
FindCategoryByNameService	Serviço que permite a pesquisa de uma categoria através do seu nome. Este serviço é utilizado para que o utilizador possa inserir o nome da categoria aquando a edição de um ponto de interesse e esse nome possa ser transformado no *ID* de categoria necessário ao registo de pontos de interesse.	35
FindCityByNameService	Serviço que permite a pesquisa de uma cidade através do seu nome. Este serviço é utilizado para que o utilizador possa inserir o nome da cidade aquando a edição de um ponto de interesse e esse nome possa ser transformado no *ID* de cidade, necessário ao registo de pontos de interesse.	36

Tabela 22 - Serviços utilizados pela componente POI

Com estes serviços definidos, é possível proceder à implementação das funcionalidades desejadas nesta componente, na qual foram utilizadas as seguintes funções:

- GetPOIsFromBackend() - função que faz um pedido ao servidor através do caminho '/pois', para obter a lista de pontos de interesse ativos disponível na base de dados;
- GetCitiesFromBackend() - função que faz um pedido ao servidor no caminho '/cities', para obter a lista de cidades ativas disponível na base de dados;
- GetCategoriesFromBackend() - função que faz um pedido ao servidor no caminho '/categories', para obter a lista de categorias ativas disponível na base de dados;
- DeleteButtonPressed() - função que faz um pedido ao servidor no caminho '/pois/delete' e que remove, da base de dados, o ponto de interesse cujo *ID* é enviado como parâmetro do pedido com recurso ao serviço *DELETEPOIService*, anteriormente descrito;

- GetPOIByIDFromBackend() - função que faz um pedido ao servidor no caminho '/pois/description', enviando o *ID* do ponto de interesse do qual se pretendem obter os dados como parâmetro do pedido;
- UpdateButtonPressed() - função que envia os novos dados relativos ao ponto de interesse que está a ser alterado para o servidor para o caminho '/pois/update', alterando, assim, os dados na base de dados, com recurso ao serviço "UpdatePOIService" atrás descrito;
- FindByName() - função que permite a pesquisa de pontos de interesse através do seu nome, com recurso ao serviço FindByNameService, anteriormente descrito;
- FindCategoryByName() - função que permite a inserção do nome da categoria aquando a edição de um ponto de interesse, sendo o mesmo transformado no *ID* da categoria, com recurso ao serviço FindCategoryByNameService, descrito anteriormente;
- FindCityByName() - função que permite a inserção do nome da cidade aquando a edição de um ponto de interesse, sendo o mesmo transformado no *ID* da cidade, com recurso ao serviço FindCityByNameService, anteriormente descrito.

Com todos estes elementos, funções e serviços descritos implementados, a componente relativa à visualização, edição e remoção de dados relativos aos pontos de interesse está completa e com todas as funcionalidades desejadas também implementadas, sendo que o seu aspeto será o que a seguir se apresenta:

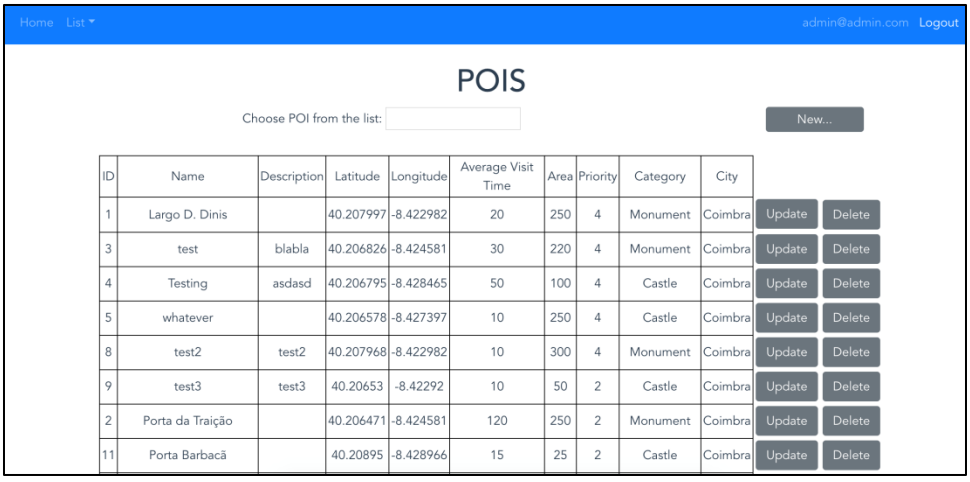


Figura 25 - Aspeto da visualização de Pontos de Interesse

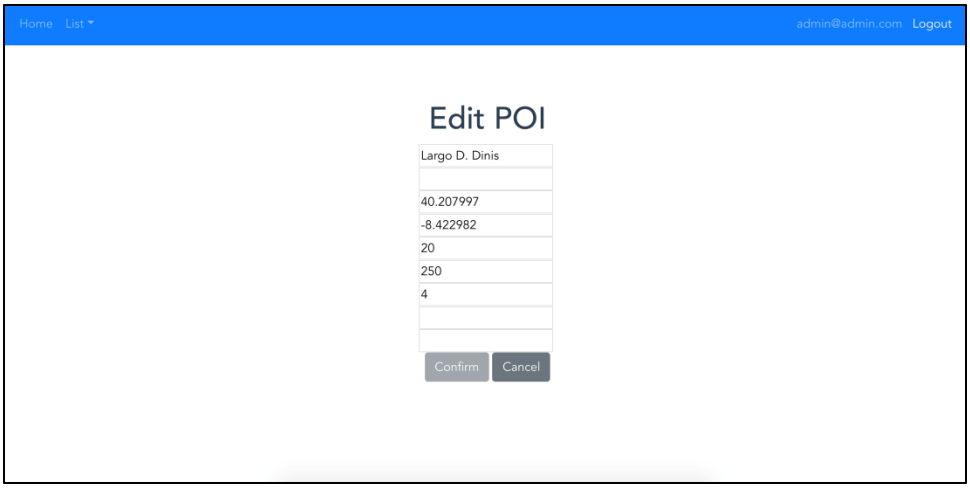


Figura 26 - Aspeto da edição de Pontos de Interesse

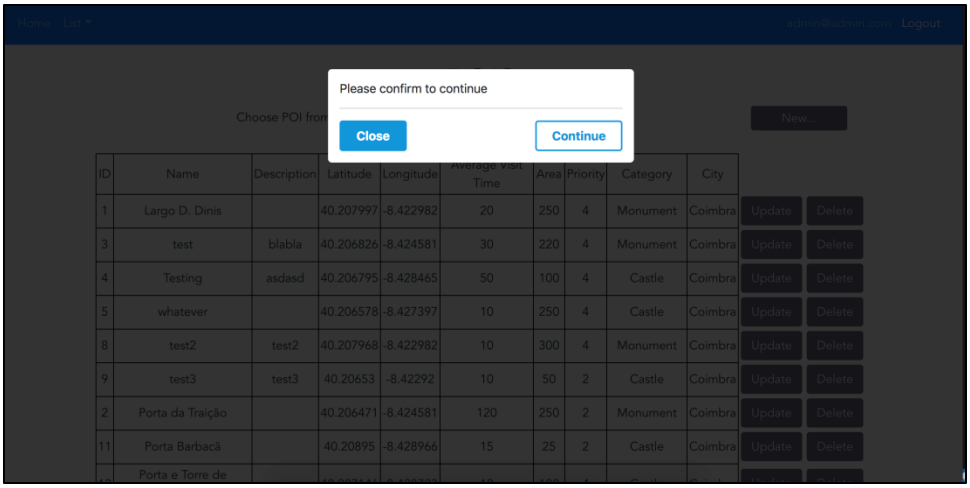


Figura 27 - Confirmação de remoção de um Ponto de Interesse

O código associado a esta componente pode ser encontrado no anexo 37.

7.7 Componente *AddPOI*

A componente *AddPOI* é aquela que permitirá a inserção de novos pontos de interesse na base de dados.

Para cumprir todos estes requisitos, esta componente é constituída pelos seguintes elementos:

- campos para introduzir nome, descrição, latitude, longitude, tempo médio de visita, área, prioridade, categoria e cidade do ponto de interesse;
- um botão para inserir o ponto de interesse na base de dados;
- um botão para cancelar a inserção e voltar à visualização dos dados relativos aos pontos de interesse.

Após os elementos a utilizar estarem definidos, foi necessária a utilização dos serviços FindByNameService, FindCategoryByNameService e "FindCityByNameService", atrás apresentados, bem como a implementação do serviço apresentado de seguida:

Serviço	Descrição	Anexo em que está apresentado
AddPOIService	Serviço que permite adicionar um ponto de interesse à base de dados, ao enviar os dados do mesmo no corpo do pedido ao caminho do servidor '/pois/add'.	38

Tabela 23 - Serviços utilizados pela componente AddPOI

Com estes serviços definidos, foi possível proceder à implementação das funcionalidades desejadas nesta componente, na qual foram utilizadas as seguintes funções:

- GetCitiesFromBackend() – função que faz um pedido ao servidor no caminho '/cities', para obter a lista de cidades ativas disponível na base de dados;
- GetCategoriesFromBackend() – função que faz um pedido ao servidor no caminho '/categories', para obter a lista de categorias ativas disponível na base de dados;
- FindCategoryByName() – função que permite a inserção do nome da categoria aquando a edição de um ponto de interesse, sendo o mesmo transformado no *ID* da categoria com recurso ao serviço "FindCategoryByNameService", anteriormente descrito;

- FindCityByName() – função que permite a inserção do nome da cidade aquando a edição de um ponto de interesse, sendo o mesmo transformado no *ID* da cidade, com recurso ao serviço "FindCityByNameService", descrito anteriormente.
- FindPOIsByCityID() – função que permite a obtenção dos dados de todos os pontos de interesse associados à cidade do novo ponto de interesse, dados esses que serão usados para a atualização da matriz de distâncias associada a cada cidade;
- GeoLongValidated() – função que valida se o valor do campo relativo à longitude é, efetivamente, um valor válido de longitude;
- GeoLatValidated() – função que valida se o valor do campo relativo à latitude é, efetivamente, um valor válido de latitude;
- SubmitValited() – função que permite a utilização do botão de inserir um novo ponto de interesse, que valida se todos os campos estão preenchidos corretamente;
- AddPOI() – função que permite a inserção de um novo ponto de interesse na base de dados, com recurso ao serviço AddPOIService, anteriormente apresentado.

Com todos estes elementos, funções e serviços já implementados, a componente relativa à inserção de dados relativos aos pontos de interesse está completa e com todas as funcionalidades desejadas implementadas, sendo que o seu aspeto será apresentado de seguida:

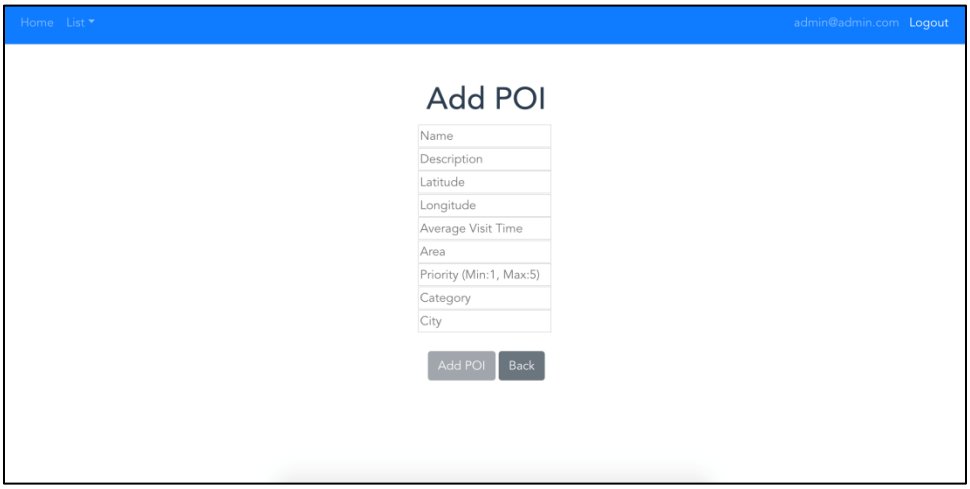


Figura 28 - Aspeto da inserção de um novo Ponto de Interesse

O código associado a esta componente pode ser encontrado no anexo 39.

7.8 Componente *Camera*

A componente *Camera* é a componente principal de interação com a tabela *Camera* da base de dados. Esta deverá permitir a visualização da informação, a edição e a remoção de dados relativos às câmaras registadas nessa mesma base de dados. Para cumprir todos estes requisitos, esta componente será constituída pelos seguintes elementos:

- uma barra de pesquisa que permita a pesquisa de câmaras através do seu endereço *IP*;
- um botão que permita a navegação para a componente *AddCamera*;
- uma tabela com que apresente a informação de cada uma das câmaras e que, nas duas últimas colunas da tabela, contenha os botões para edição e remoção das mesmas;
- no caso de se selecionar a funcionalidade de edição de uma câmara, os elementos da componente inicialmente apresentados ao utilizador devem ser escondidos e deverão ser apresentados diversos elementos para a inserção dos diferentes atributos da câmara, que devem ser preenchidos automaticamente com a informação existente da mesma, para que possam ser alterados pelo utilizador, conforme seja do seu agrado;
- no caso de se selecionar a funcionalidade de remoção de uma câmara, deve ser apresentado um aviso em que o utilizador deve confirmar ou cancelar a remoção do ponto de interesse.

Após os elementos a utilizar estarem definidos, foi necessária a implementação dos serviços:

Serviço	**Descrição**	**Anexo em que está apresentado**
FindByIPService	Serviço que permite a pesquisa de uma câmara através do seu endereço *IP*, enviando o conteúdo do elemento barra de pesquisa, mencionado anteriormente, no corpo do pedido ao caminho do servidor '/cameras/findbyip'.	40
UpdateCameraService	Serviço que permite a alteração dos dados de uma câmara, que deverá esconder todos os elementos inicialmente mostrados ao utilizador e mostrar os elementos de inserção de dados relativos à câmara escolhida, com recurso ao caminho do servidor '/cameras/update'.	41

DeleteCameraService	Serviço que permite a remoção dos dados da câmara cujo *ID* é enviado como parâmetro do pedido, com recurso ao caminho do servidor '/cameras/delete'	42

Tabela 24 - Serviços utilizados pela componente Camera

Para além da implementação destes novos serviços, foi, também, necessária a utilização do serviço FindByNameService apresentado na componente *POI*.

Com estes serviços definidos, foi possível proceder à implementação das funcionalidades desejadas nesta componente, na qual foram utilizadas as seguintes funções:

- GetPOIsFromBackend() – função que faz um pedido ao servidor no caminho '/pois', para obter a lista de pontos de interesse ativos disponível na base de dados;
- GetCamerasFromBackend() – função que faz um pedido ao servidor no caminho '/cameras', para obter a lista de câmaras ativas disponível na base de dados;
- GetCameraByIDFromBackend() – função que faz um pedido ao servidor no caminho '/cameras/description', enviando o *ID* da câmara do qual se pretendem obter os dados como parâmetro do pedido;
- DeleteButtonPressed() – função que faz um pedido ao servidor no caminho '/cameras/delete' e que remove da base de dados a câmara cujo *ID* é enviado como parâmetro do pedido com recurso ao serviço DeleteCameraService, atrás descrito;
- UpdateButtonPressed() – função que envia os novos dados relativos à câmara que está a ser alterada para o servidor pelo caminho '/camera/update', alterando, assim, os dados na base de dados, com recurso ao serviço UpdateCameraService, anteriormente descrito;
- FindByIP() – função que permite a pesquisa de câmaras através do seu endereço *IP*, com recurso ao serviço FindByIPService, anteriormente descrito.

Com todos estes elementos, funções e serviços já descritos implementados, a componente relativa à visualização, edição e remoção de dados respeitantes às câmaras está completa e com todas as funcionalidades desejadas implementadas, sendo que o seu aspeto será o que a seguir se apresenta:

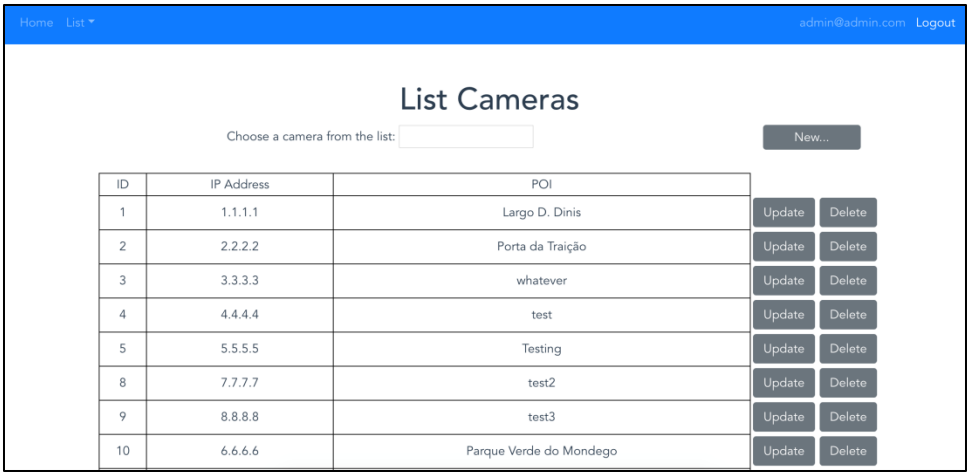


Figura 29 - Aspeto da visualização de câmaras

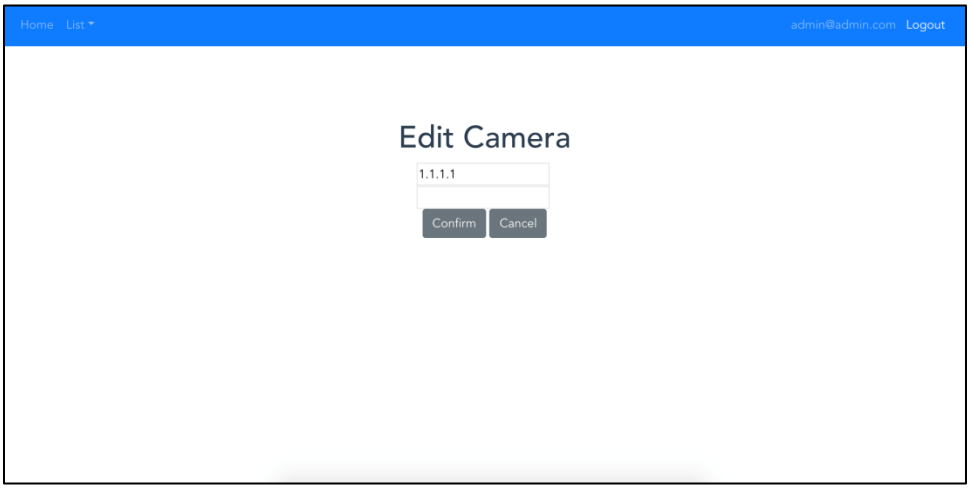


Figura 30 - Aspeto da edição de câmaras

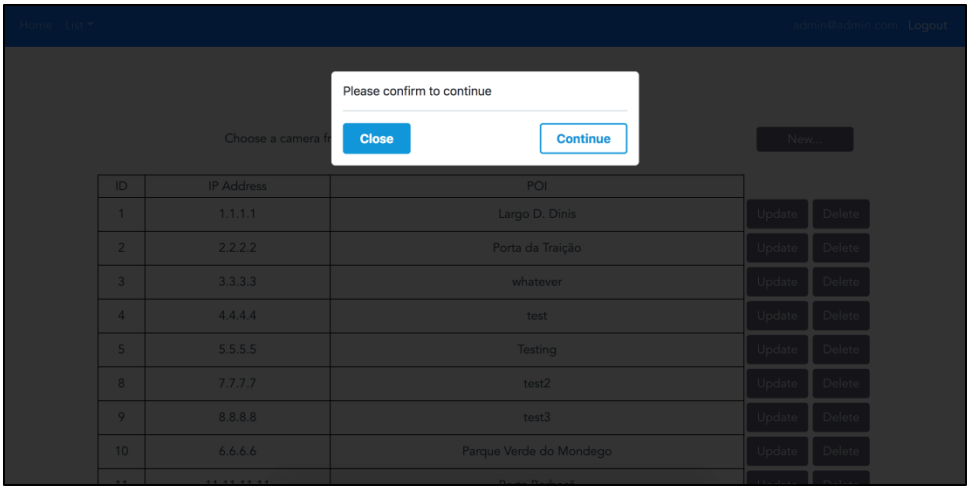


Figura 31 - Confirmação de remoção de uma câmara

O código associado a esta componente pode ser encontrado no anexo 43.

7.9 Componente *AddCamera*

A componente AddCamera será a componente que permitirá a inserção de uma nova câmara na base de dados. Para cumprir todos estes requisitos, esta componente será constituída pelos seguintes elementos:

- Campos para introduzir o endereço IP da câmara e o ponto de interesse associado à mesma;
- Um botão para inserir a câmara na base de dados;
- Um botão para cancelar a inserção e voltar à visualização dos dados relativos às câmaras.

Após os elementos a utilizar estarem definidos, foi necessária a utilização do serviço "FindByNameService" apresentado anteriormente, bem como a implementação do serviço apresentado de seguida:

Serviço	**Descrição**	**Anexo em que está apresentado**
AddCameraService	Serviço que permite adicionar uma câmara à base de dados ao enviar os	44

	dados da mesma no corpo do pedido ao caminho do servidor '/cameras/add'	

Tabela 25 - Serviços utilizados pela componente AddCamera

Com estes serviços definidos, é possível proceder à implementação das funcionalidades desejadas nesta componente, na qual foram utilizadas as seguintes funções:

- FindByName() - função que permite a pesquisa de pontos de interesse através do seu nome para associação da câmara ao mesmo;
- addCamera() - função que permite a inserção de uma nova câmara na base de dados com recurso ao serviço "AddCameraService" descrito anteriormente;
- ValidIP() - função que valida se o endereço IP inserido pelo utilizador é válido

Com todos estes elementos, funções e serviços descritos anteriormente implementados, a componente relativa à inserção de dados relativos aos pontos de interesse está completa e com todas as funcionalidades desejadas implementadas, sendo que o seu aspeto será apresentado de seguida:

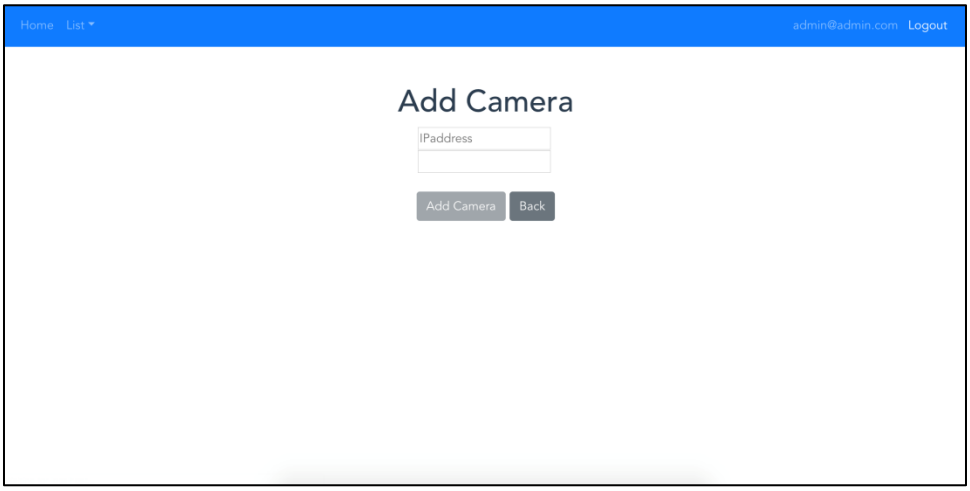


Figura 32 - Aspeto da inserção de uma nova câmara

O código associado a esta componente pode ser encontrado no anexo 45.

7.10 Componente *Category*

A componente Categoria será a componente de interação com a tabela Category da base de dados. Esta deverá, tal como os componentes anteriores, permitir a visualização de todas as categorias, a pesquisa de categorias através do seu nome, a edição e remoção de categorias de bases de dados, bem como deve conter um botão de acesso à componente AddCategory descrita anteriormente. Para cumprir todos estes requisitos, esta componente será constituída pelos seguintes elementos:

- Uma barra de pesquisa que permita a pesquisa de categorias pelo seu nome;
- Um botão que permita a navegação para a componente AddCategory;
- Uma tabela com que apresente a informação de cada uma das categorias e que, nas duas últimas colunas da tabela, contenha os botões para edição e remoção de uma categoria;
- No caso de se selecionar a funcionalidade de edição de uma categoria, os elementos da componente apresentados inicialmente ao utilizador devem ser escondidos e deverão ser apresentados diversos elementos para a inserção dos diversos atributos da categoria, que devem ser preenchidos automaticamente com a informação existente da categoria para que possam ser alterados pelo utilizador ao seu agrado;
- No caso de se selecionar a funcionalidade de remoção de uma categoria deve ser apresentado um aviso em que o utilizador deve confirmar ou cancelar a remoção da mesma.

Após os elementos a utilizar estarem definidos, foi necessária a implementação dos serviços:

Serviço	**Descrição**	**Anexo em que está apresentado**
FindCategoryByNameService	Serviço que permite a pesquisa de uma categoria através do seu nome, enviando o conteúdo do elemento barra de pesquisa mencionado anteriormente no URL do pedido ao caminho do servidor '/pois/categories/findbyname'	35
UpdateCategoryService	Serviço que permite a alteração dos dados de uma cidade, que deverá esconder todos os elementos mostrados inicialmente ao utilizador e mostrar os elementos de inserção de dados relativos a uma categoria, com recurso ao caminho do servidor '/pois/categories/update'	46
DeleteCategoryService	Serviço que permite a remoção dos dados da categoria cujo ID é enviado como parâmetro do pedido, com	47

	recurso ao caminho do servidor '/pois/categories/delete'	

Tabela 26 - Serviços utilizados pela componente Category

Com estes serviços definidos, é possível proceder à implementação das funcionalidades desejadas nesta componente, na qual foram utilizadas as seguintes funções:

- GetCategoriesFromBackend() - função que faz um pedido ao servidor no caminho '/pois/categories' para obter a lista de categorias ativas disponível na base de dados;
- DeleteButtonPressed() - função que faz um pedido ao servidor no caminho '/pois/categories/delete' e que remove da base de dados a categoria cujo ID é enviado como parâmetro do pedido com recurso ao serviço "DeleteCategoryService" descrito anteriormente;
- UpdateButtonPressed() - função que envia os novos dados relativos à categoria que está a ser alterada para o servidor para o caminho '/pois/categories/update', alterando assim os dados na base de dados com recurso ao serviço "UpdateCategoryService" descrito anteriormente;
- FindCategoryByName() - função que permite a inserção do nome da categoria aquando a edição de uma categoria, sendo o mesmo transformado no ID da categoria com recurso ao serviço "FindCategoryByNameService" descrito anteriormente;
- FindCategoryByIDFromBackend() - função que permite a obtenção dos dados relativos à categoria cujo ID é enviado como parâmetro, para poder preencher automaticamente os campos da categoria a alterar com os dados já existentes na base de dados.

Com todos estes elementos, funções e serviços descritos anteriormente implementados, a componente relativa à visualização, edição e remoção de dados relativos às categorias está completa e com todas as funcionalidades desejadas implementadas, sendo que o seu aspeto será apresentado de seguida:

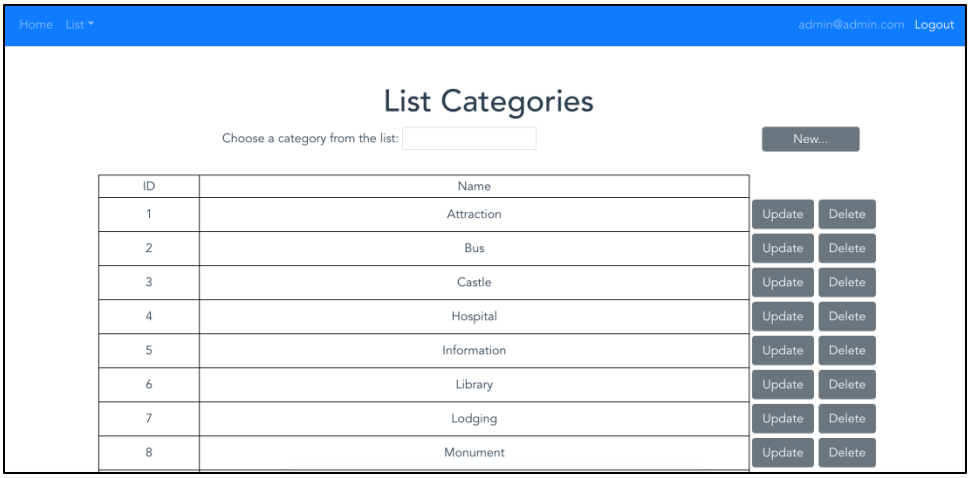


Figura 33 - Aspeto da visualização de categorias

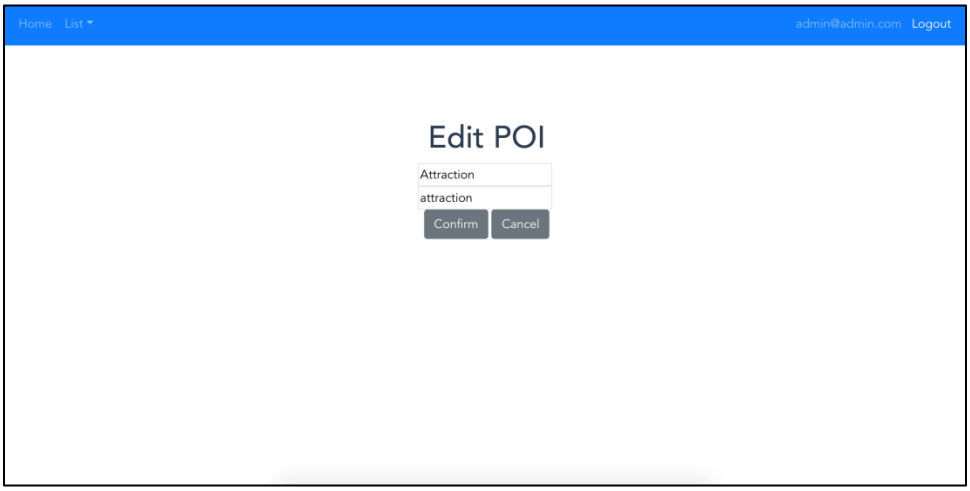


Figura 34 - Aspeto da edição de categorias

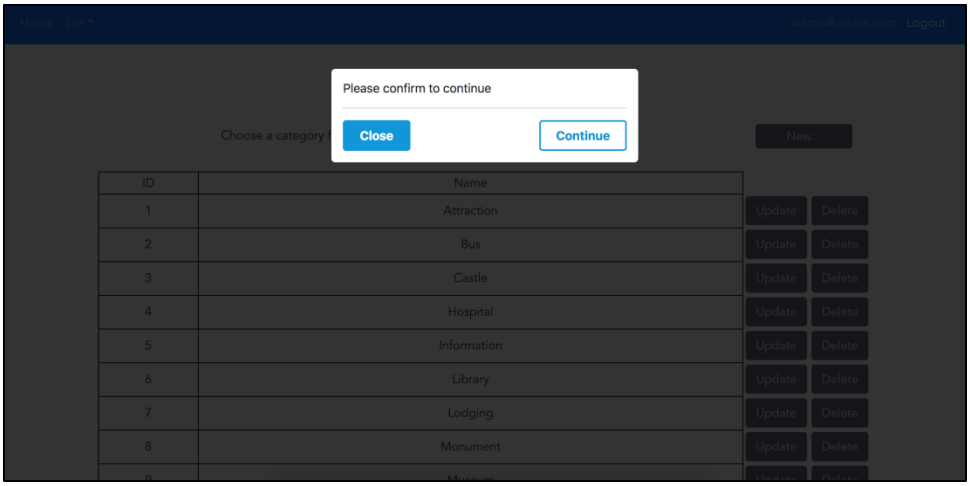


Figura 35 - Confirmação da remoção de uma categoria

O código associado a esta componente pode ser encontrado no anexo 48.

7.11 Componente *AddCategory*

A componente AddCategory será a componente que permitirá a inserção de uma nova categoria na base de dados. Para cumprir todos estes requisitos, esta componente será constituída pelos seguintes elementos:

- Campos para introduzir o nome da categoria e o nome do ícone a apresentar no mapa associado à mesma;
- Um botão para inserir a categoria na base de dados;
- Um botão para cancelar a inserção e voltar à visualização dos dados relativos às categorias.

Tendo em conta que a tabela categoria da base de dados é uma tabela que não apresenta nenhuma chave estrangeira na mesma, apenas foi necessária a criação serviço apresentado de seguida:

Serviço	**Descrição**	**Anexo em que está apresentado**
AddCategoryService	Serviço que permite adicionar uma categoria à base de dados ao enviar os dados do mesmo no corpo do pedido ao caminho do servidor	49

	'/pois/categories/add'	

Tabela 27 - Serviços utilizados pela componente AddCategory

Para a implementação da funcionalidade descrita anteriormente para a componente a ser descrita, foi apenas criada a função AddCategory() que, com recurso ao serviço apresentado na tabela anterior, permite a inserção de uma nova categoria na base de dados.

Com todos estes elementos, funções e serviços descritos anteriormente implementados, a componente relativa à inserção de dados relativos às categorias está completa e com todas as funcionalidades desejadas implementadas, sendo que o seu aspeto será apresentado de seguida:

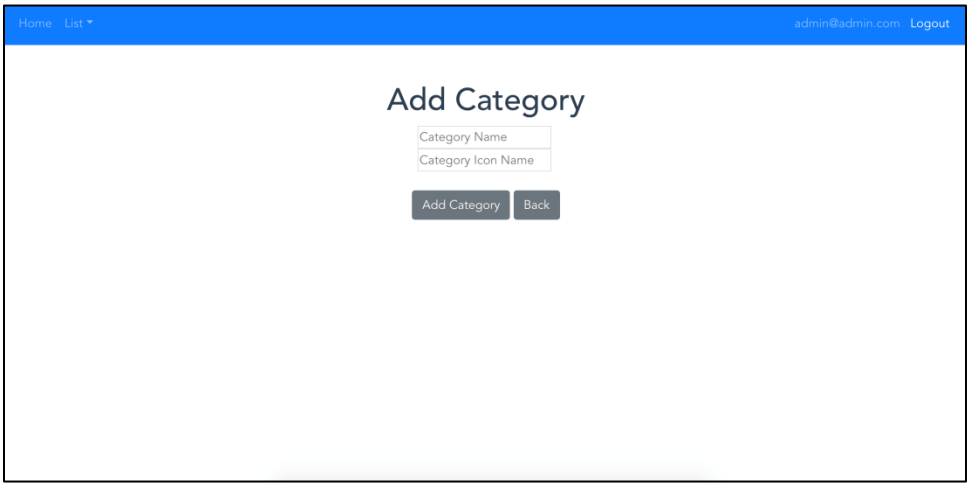


Figura 36 - Aspeto da inserção de uma nova categoria

O código associado a esta componente pode ser encontrado no anexo 50.

7.12 Componente *City*

A componente City será a componente de interação com a tabela City da base de dados. Esta deverá, tal como os componentes anteriores, permitir a visualização de todas as cidades, a pesquisa de cidades através do seu nome, a edição e remoção de cidades da base de dados, bem como deve conter um botão de acesso à componente AddCity . Para cumprir todos estes requisitos, esta componente será constituída pelos seguintes elementos:

- Uma barra de pesquisa que permita a pesquisa de cidades pelo seu nome;
- Um botão que permita a navegação para a componente AddCity;
- Uma tabela com que apresente a informação de cada uma das cidades e que, nas duas últimas colunas da tabela, contenha os botões para edição e remoção de uma cidade;

- No caso de se selecionar a funcionalidade de edição de uma cidade, os elementos da componente apresentados inicialmente ao utilizador devem ser escondidos e deverão ser apresentados diversos elementos para a inserção dos diversos atributos da cidade, que devem ser preenchidos automaticamente com a informação existente da mesma para que possam ser alterados pelo utilizador ao seu agrado;
- No caso de se selecionar a funcionalidade de remoção de uma cidade deve ser apresentado um aviso em que o utilizador deve confirmar ou cancelar a remoção da mesma.

Após os elementos a utilizar estarem definidos, foi necessária a implementação dos serviços:

Serviço	Descrição	Anexo em que está apresentado
FindCityByNameService	Serviço que permite a pesquisa de uma cidade através do seu nome, enviando o conteúdo do elemento barra de pesquisa como argumento do pedido ao caminho do servidor '/cities/findbyname'	36
UpdateCityService	Serviço que permite a alteração dos dados de uma cidade, que deverá esconder todos os elementos mostrados inicialmente ao utilizador e mostrar os elementos de inserção de dados relativos a uma cidade, com recurso ao caminho do servidor '/cities/update'	51
DeleteCityService	Serviço que permite a remoção dos dados da cidade cujo ID é enviado como parâmetro do pedido, com recurso ao caminho do servidor '/cities/delete'	52

Tabela 28 - Serviços utilizados pela componente City

Com estes serviços definidos, é possível proceder à implementação das funcionalidades desejadas nesta componente, na qual foram utilizadas as seguintes funções:

- GetCitiesFromBackend() - função que faz um pedido ao servidor no caminho '/cities' para obter a lista de cidades ativas disponível na base de dados;
- DeleteButtonPressed() - função que faz um pedido ao servidor no caminho '/cities/delete' e que remove da base de dados a cidade cujo ID é enviado como parâmetro do pedido com recurso ao serviço "DeleteCityService" descrito anteriormente;

- UpdateButtonPressed() - função que envia os novos dados relativos à cidade que está a ser alterada para o servidor para o caminho '/cities/update', alterando assim os dados na base de dados com recurso ao serviço "UpdateCityService" descrito anteriormente;
- FindCityByName() - função que permite a inserção do nome da cidade aquando a edição de um ponto de interesse, sendo o mesmo transformado no ID da cidade com recurso ao serviço "FindCityByNameService";
- FindCityByIDFromBackend() - função que permite a obtenção dos dados relativos à cidade cujo ID é enviado como parâmetro, para poder preencher automaticamente os campos da cidade a alterar com os dados já existentes na base de dados.

Com todos estes elementos, funções e serviços descritos anteriormente implementados, a componente relativa à visualização, edição e remoção de dados relativos às cidades está completa e com todas as funcionalidades desejadas implementadas, sendo que o seu aspeto será apresentado de seguida:

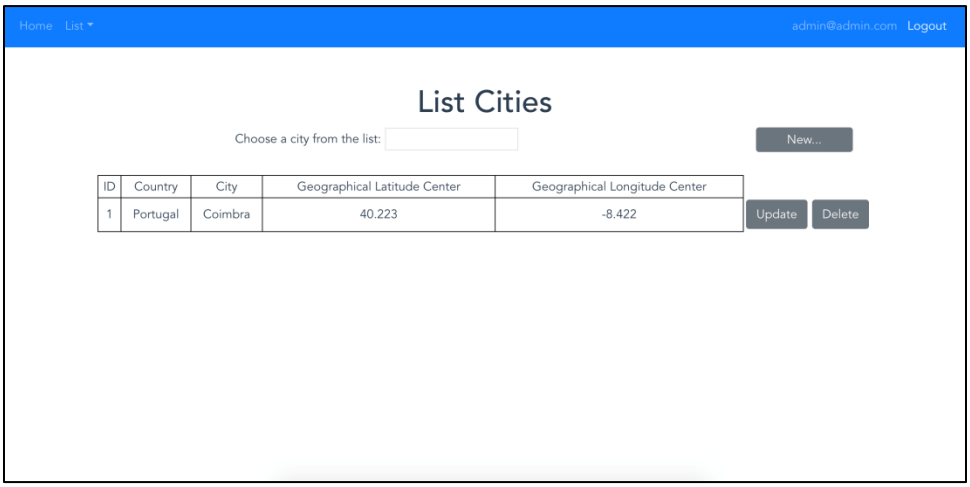


Figura 37 - Aspeto da visualização de cidades

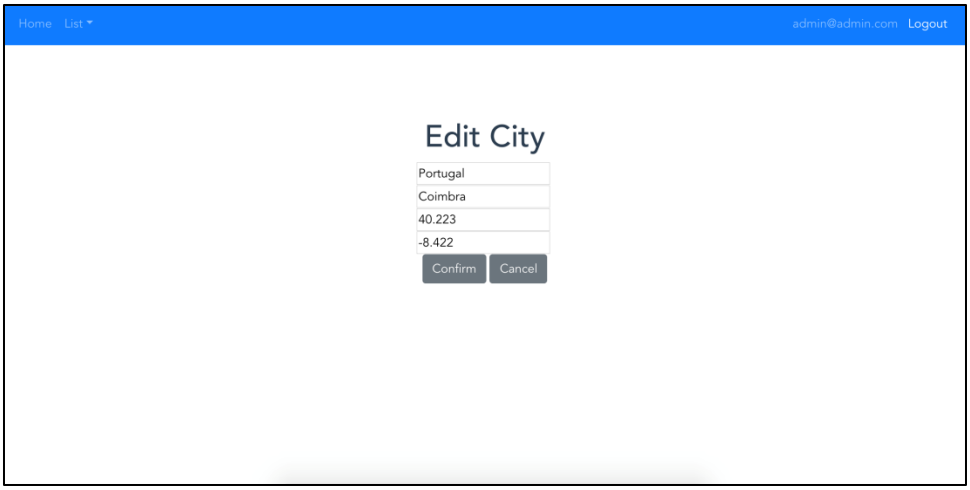


Figura 38 - Aspeto da edição de cidades

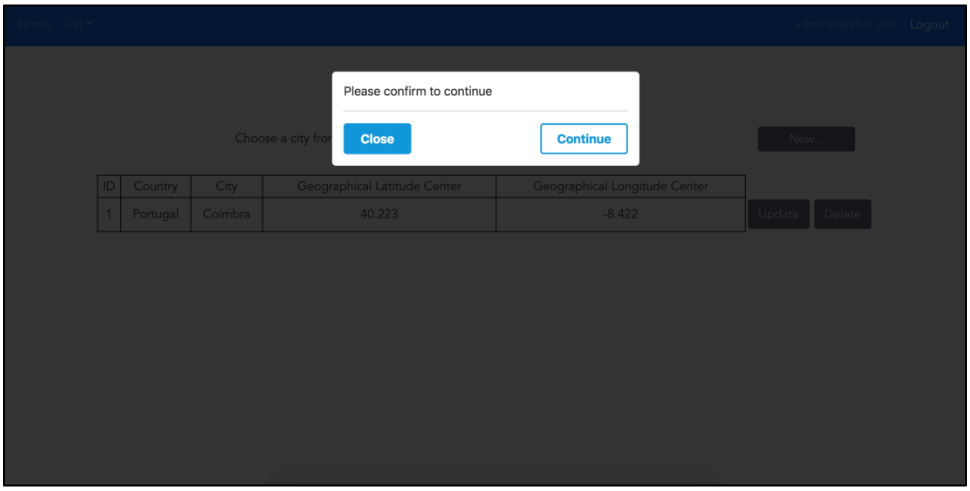


Figura 39 - Confirmação da remoção de uma cidade

O código associado a esta componente pode ser encontrado no anexo 53.

7.13 Componente *AddCity*

A componente AddCity será a componente que permitirá a inserção de uma nova cidade na base de dados. Para cumprir todos estes requisitos, esta componente será constituída pelos seguintes elementos:

- Campos para introduzir o nome do país a que pertence a cidade, o nome da cidade e o seu centro geográfico de latitude e de longitude necessários para centrar a cidade quando a mesma for apresentada no mapa;
- Um botão para inserir a cidade na base de dados;
- Um botão para cancelar a inserção e voltar à visualização dos dados relativos às cidades.

Tendo em conta que a tabela City da base de dados é uma tabela que não apresenta nenhuma chave estrangeira na mesma, apenas foi necessária a criação serviço apresentado de seguida:

Serviço	**Descrição**	**Anexo em que está apresentado**
AddCityService	Serviço que permite adicionar uma cidade à base de dados ao enviar os dados da mesma no corpo do pedido ao caminho do servidor '/cities/add'	54

Tabela 29 - Serviços utilizados pela componente AddCity

Com estes serviços definidos, é possível proceder à implementação das funcionalidades desejadas nesta componente, na qual foram utilizadas as seguintes funções:

- AddCity() - função que faz um pedido ao servidor no caminho '/cities/add' para obter a lista de categorias ativas disponível na base de dados;
- GeoLatValidated() - função que verifica se o valor de latitude inserido é válido;
- GeoLongValidated() - função que verifica se o valor de longitude inserido é válido;
- SubmitValidated() - função que permite a utilização do botão de inserção da cidade na base de dados quando ambas a latitude e longitude forem válidas

Com todos estes elementos, funções e serviços descritos anteriormente implementados, a componente relativa à inserção de dados relativos às categorias está completa e com todas as funcionalidades desejadas implementadas, sendo que o seu aspeto será apresentado de seguida:

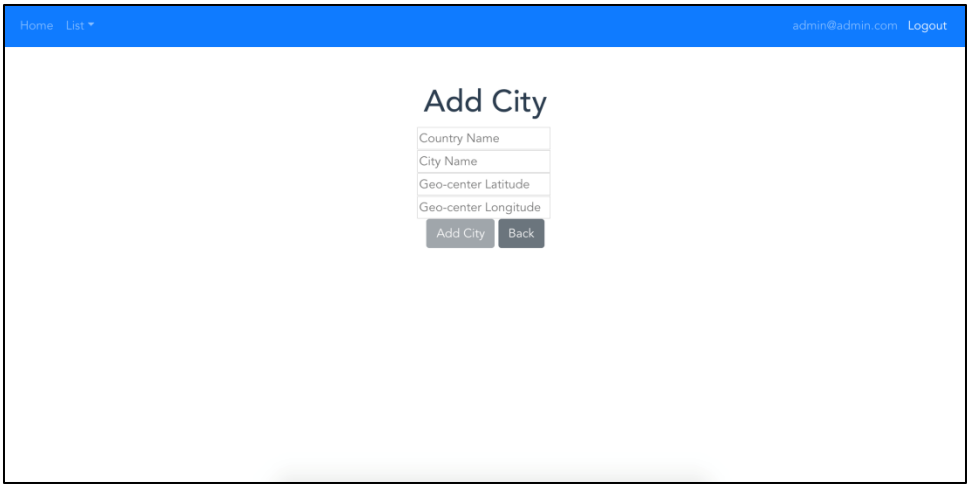


Figura 40 - Aspeto da inserção de uma nova cidade

O código associado a esta componente pode ser encontrado no anexo 55.

7.14 Componente *Prediction*

A componente Prediction será a componente de interação com a tabela Prediction da base de dados, a tabela que guarda toda a informação relacionada com as classificações que os modelos apresentados no capítulo 5.3 .

Tendo em conta o funcionamento geral do funcionamento da aplicação proposto na figura XXXX (esquema de funcionamento das câmaras), o utilizador deverá apenas ter acesso à informação da base de dados e não terá acesso aos caminhos do servidor para adicionar, editar nem remover os dados da base de dados. Em vez disso, o script de classificação dos vídeos será o único a inserir e editar dados relativos às previsões.

De forma a promover o registo histórico das classificações de taxas de ocupação e com o intuito de, como implementações futuras, aconselhar o utilizador a visitar pontos de interesse baseado na análise histórica do mesmo, a remoção dos dados relativos a previsões na base de dados não foi implementada.

Como tal, esta componente será apenas constituída por uma tabela que irá conter os dados de todas as previsões para o utilizador visualizar, tal como se pode verificar na seguinte figura:

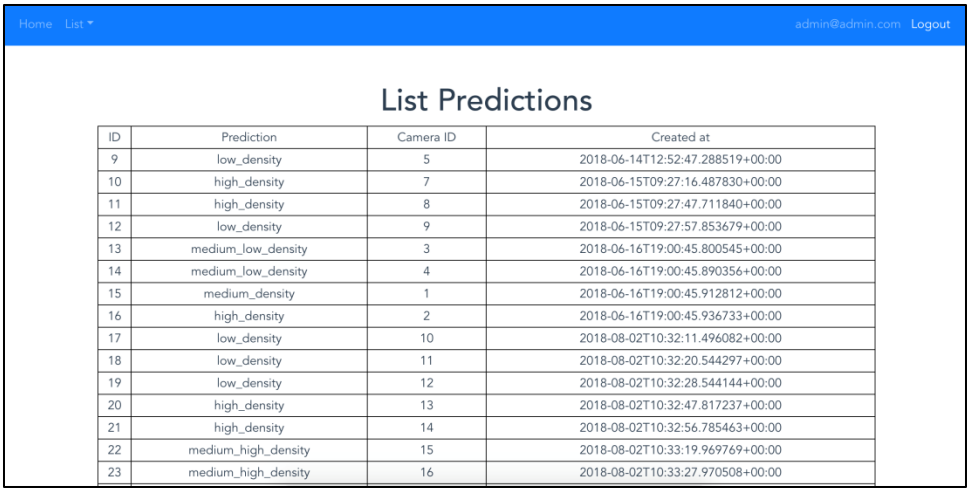


Figura 41 - Aspeto da visualização de previsões

Para tornar a visualização dos dados relativos às previsões possível foi necessário a implementação da função "getPredictionsFromBackend()" que faz um pedido ao servidor na rota '/predictions' e retorna todas as previsões ativas.

O código final deste componente pode ser visualizado no anexo 56.

7.15 Componente *User*

A componente User será a componente que permitirá a interação do utilizador com a tabela da base de dados que contém toda a informação relativa ao utilizador.

Tendo em conta que a criação de novos utilizadores é feita pelos mesmos, na componente Register descrita anteriormente, a componente User deverá permitir a visualização da informação de todos os utilizadores, a pesquisa de utilizadores por email, bem como deve possuir um botão que permita editar um utilizador, que neste caso será a edição apenas do atributo "role", ou a remoção completa dos dados de um utilizador (para poder cumprir o RGPD).

Quando o utilizador seleciona o botão de edição do "role" de um utilizador, todos os elementos definidos anteriormente devem ser escondidos e deverá ser tornado visível um campo para a inserção do novo "role" do utilizador, que terá um botão de confirmação e cancelamento associados

Tendo a componente definida, o seu aspeto final, tanto quando a página é carregada como quando o utilizador tenta alterar o "role" de um outro utilizador, é apresentado nas seguintes figuras:

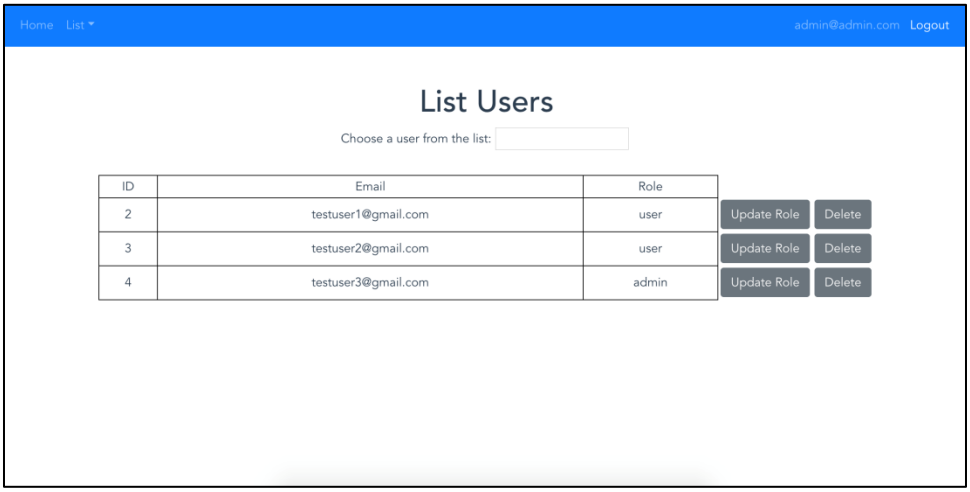


Figura 42 - Aspeto da visualização de utilizadores

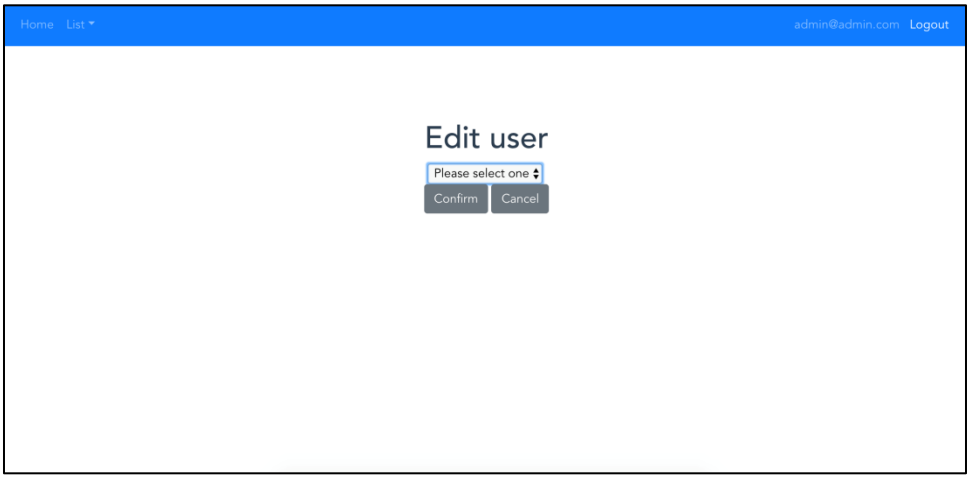


Figura 43 - Aspeto da edição do tipo de utilizador

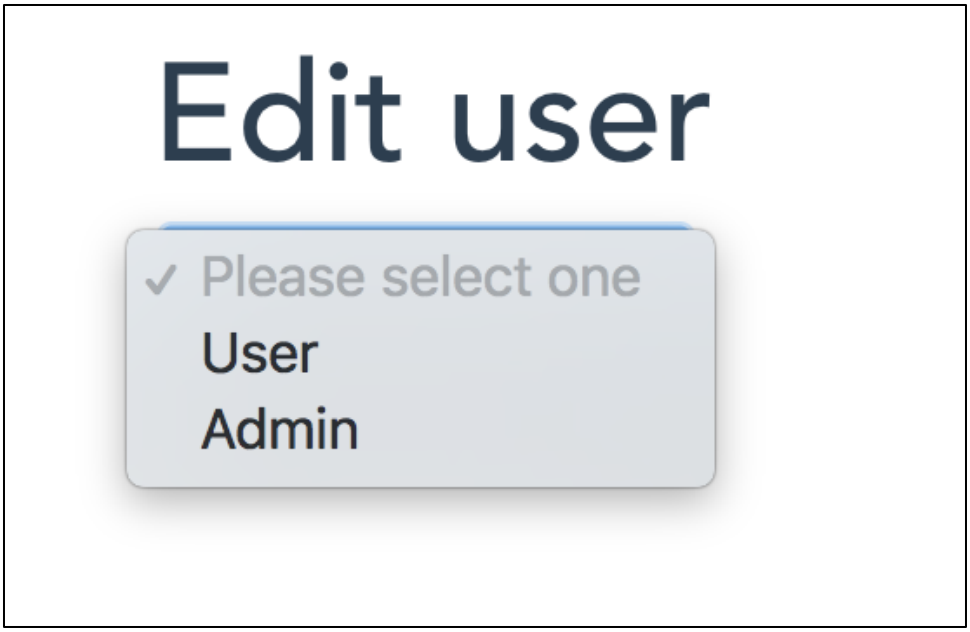


Figura 44 - Escolhas possíveis para tipos de utilizador

O código associado a esta componente pode ser encontrado no anexo 57.

7.16 Componente Route

A componente Route será a componente em que serão implementadas as funcionalidades de criação de rotas e de visualização das mesmas num mapa.

Nesta componente o utilizador irá, inicialmente, poder visualizar um menu de criação de rotas, cujo primeiro parâmetro é a cidade na qual se deseja criar a rota e que é apresentado do lado esquerdo da página, e um mapa criado utilizando a biblioteca Mapbox GL JS, que ocupa a maior parte do ecrã e permite ao utilizador interagir com o mapa livremente, tal como se pode verificar na figura seguinte:

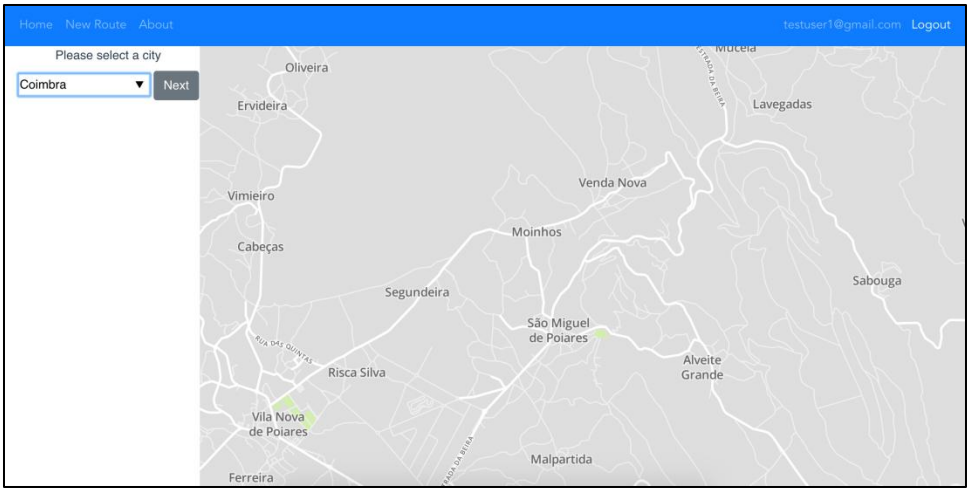


Figura 45 - Menu de escolha de cidade da component Route

O menu lateral de criação de rotas irá guiar o utilizador através do processo de seleção da cidade na qual o utilizador pretende criar a rota, de seleção do tipo de criação de rota, sendo que os dois tipos possíveis são o tipo personalizado e o tipo automático baseado na duração inserida pelo utilizador descritos no capítulo anterior de Modelos e Algoritmos de Rotas.

A apresentação do menu para a seleção do tipo de funcionamento e apresentada de seguida:

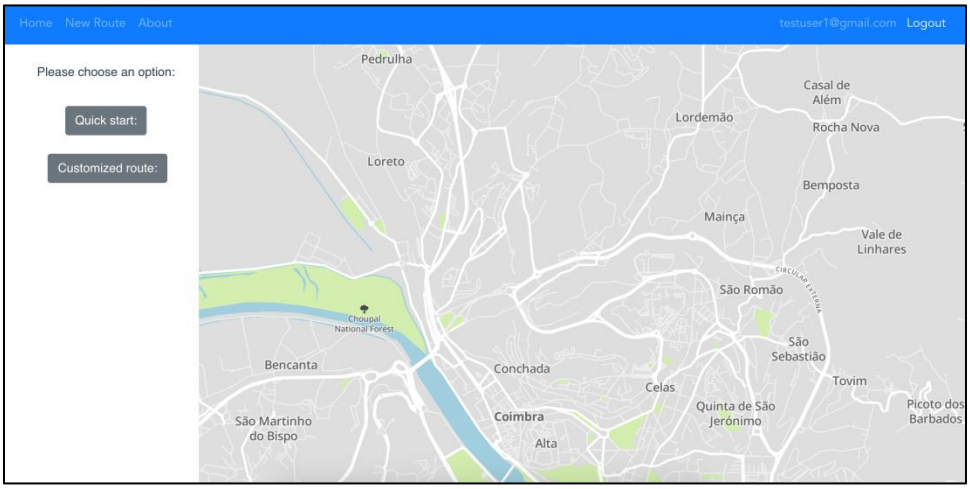


Figura 46 - Menu de escolha do modo de funcionamento da componente Route

No caso de o utilizador selecionar o modo de criação de rotas "Quick start", é-lhe apresentado um campo para inserir a duração desejada para a rota a criar e um botão para confirmar a inserção do valor.

Assim que o utilizador seleciona o botão de confirmação, o algoritmo de rotas automático baseado na duração inserida pelo utilizador é executado e a rota obtida pelo mesmo é apresentada ao utilizador no mapa.

Assim é possível obter a rota criada, tal como se pode verificar na seguinte figura:

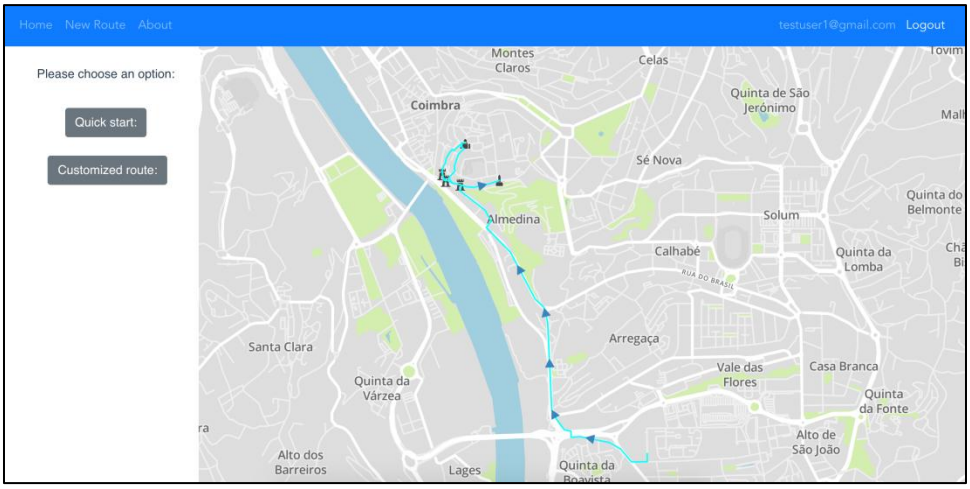


Figura 47 - Exemplo de uma rota criada pelo modo automático

No caso de o utilizador selecionar o modo de criação de rotas "Customized route" no menu apresentado na figura 45, é-lhe apresentada uma lista de categorias, da qual o utilizador pode selecionar quais as categorias de pontos de interesse que o mesmo tem interesse em visitar. Este menu de seleção de categorias de interesse pode ser visualizado na seguinte figura:

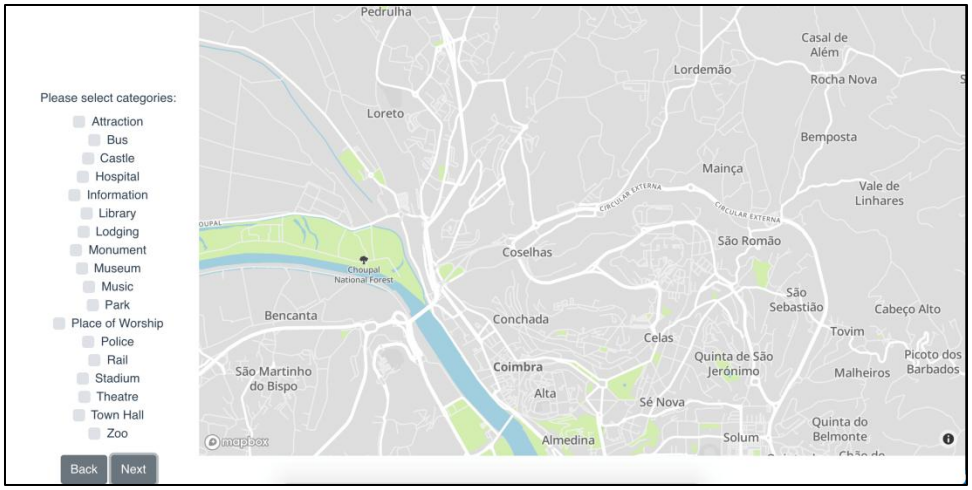


Figura 48 - Menu de escolha de categorias de interesse para o modo de funcionamento personalizado da componente Route

Assim que o utilizador seleciona uma ou mais categorias e confirma a sua escolha, é-lhe apresentada a lista de todos os pontos de interesse pertencentes à cidade e às categorias definidas anteriormente, sendo os mesmos apresentados no mapa com um ícone que caracterize a categoria do mesmo, com um "pop-up" implementado para permitir a visualização do nome do ponto de interesse a que pertence o ícone selecionado. O utilizador deve também, neste ponto do processo de criação da rota, selecionar o primeiro e o último pontos de interesse da rota.

Este funcionamento pode ser visualizado na seguinte figura:

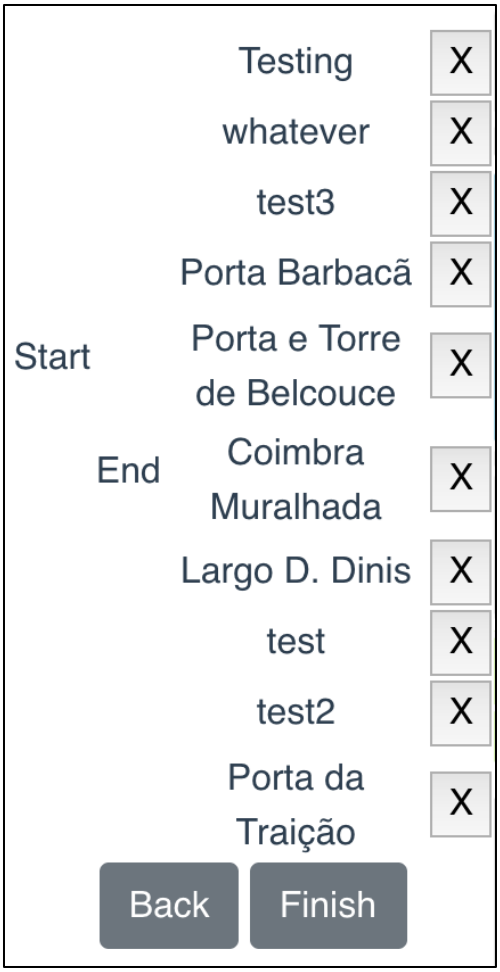


Figura 49 - Menu de escolha de pontos de interesse a visitar e de início e fim da rota

Com a apresentação dos pontos de interesse no mapa, o intuito é que o utilizador possa ter acesso ao nome e descrição do ponto de interesse no "pop-up" implementado no mesmo, para o mesmo poder decidir se quer que aquele ponto de interesse seja incluído na rota e, se não, removê-lo.

Assim que o utilizador seleciona o botão de confirmação, o algoritmo de rotas automático baseado na duração inserida pelo utilizador é executado e a rota obtida pelo mesmo é apresentada ao utilizador no mapa.

Apresenta-se de seguida um exemplo de uma rota criada pelo algoritmo a descrito:

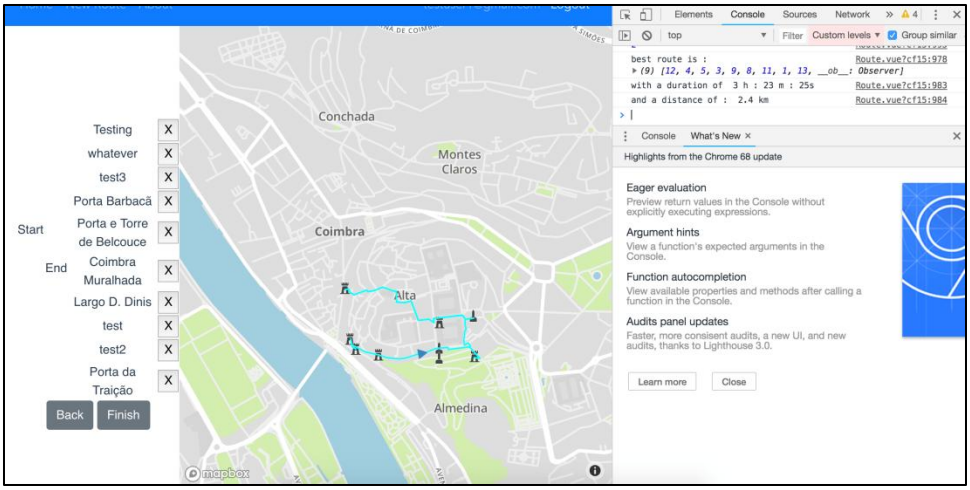


Figura 50 - Menu de apresentação da rota criada pelo modo personalizado

Assim, o utilizador poderá criar uma rota completamente personalizada, baseada nas suas categorias de interesse e que, idealmente, irá satisfazer o utilizador.

O código associado a esta componente pode ser encontrado no anexo 58.

8 Conclusão

O trabalho aqui descrito procura ser inovador, criativo e útil, três coisas que nos parecem fundamentais, seja em termos tecnológicos, seja em termos de vida atual.

Com efeito, estamos todos um pouco cansados do "mais do mesmo", isto é, de aplicações e de sites que têm a mesma função e/ou que nos remetem para as mesmas coisas. Daí que a inovação, nas novas tecnologias como na vida, devam sempre ser um objetivo e devam, consequentemente, ser valorizadas. Neste caso em particular, quisemos fugir a esse tal "mais do mesmo" e apostámos numa plataforma inovadora, capaz de oferecer, ao seu utilizador-alvo, as potencialidades e as funcionalidades que ele procura. Porém, para além da inovação, procurámos não descurar outros fatores, como a "simplicidade" (procurámos desenvolver de uma plataforma de utilização simples e quase intuitiva) e a segurança (no mundo atual, o utilizador tem de sentir que as ofertas que tem à sua disposição são fiáveis e que os dados que nelas introduzem estão abrangidos por códigos de segurança que cumprem, escrupulosamente, as regras vigentes (remetemos, nomeadamente, para o atrás mencionado no que à política de proteção de dados diz respeito).

Quanto à aplicação que nos foi solicitado desenvolver, vimo-la como um desafio: um desafio às competências técnica e de planeamento; um desafio à capacidade de inovar (qualquer aplicação que se pretenda de sucesso tem que ser eficaz, mas também inovadora); um desafio à gestão do tempo; um desafio à aprendizagem constante e contínua, que qualquer programador, como qualquer outro profissional, seja lá de que área for, deve sentir; um desafio, enfim, de grande envergadura.

Mas são os desafios que nos fazem progredir, melhorar, avançar, pelo que foi, desde o início, um desafio aceite, um projeto cuja camisola "vestimos" desde o primeiro minuto.

As dificuldades que surgiam fizeram-nos, claramente, perder muitas horas de sono a tentar encontrar soluções. Mas era o tal desafio que guiava o comportamento, que dava energias, que levava à pesquisa, ao treino, às experiências, às tentativas, a tudo o que, no final, permitisse alcançar o sucesso desejado.

Ao longo do seu desenvolvimento, foram alguns os problemas encontrados (fundamentalmente ao nível da captação de imagens e do seu uso), mas, com bom-senso, com cuidado, com muito trabalho de "bastidores" e com a colaboração de Humberto Rendeiro do Museu Monográfico e Ruínas de Conimbriga, pudemos encontrar uma solução que nos permitiu contornar esse problema e verificar que a aplicação criada efetivamente funciona.

Outro dos problemas com o qual tivemos que lidar foi o trabalhar a imagem de forma a que os elementos que a integravam (monumentos, carros, pessoas, cães…) pudessem ser reconhecidos como tal. De facto, Peter Steiner dizia ao jornal *The New Yorker*, em 1993, que, "na internet, ninguém sabe que você é cão"; claramente não se referiria ao trabalho por nós desenvolvido e aqui apresentado, mas, curiosamente, esse foi um dos problemas mais difíceis de ultrapassar: conseguir identificar cada elemento da imagem como aquilo que era efetivamente. A partir do mento em que conseguimos fazê-lo, as potencialidades da aplicação tornaram-se maiores, já que a sua eficácia aumentou tremendamente: quando procuramos saber o nível de frequência de um determinado local, a aplicação diz-nos, realmente, se há, ou não, muita gente, isto é, se o local está, ou não, com demasiada "densidade populacional" para os nossos intentos e, sobretudo, para o tempo que temos disponível, oferecendo-nos outra(s) possibilidade(s) de percurso, com o intuito de nos fazer despender o mínimo de tempo possível, seja entre os diferentes lugares do percurso a fazer, seja na visita a cada um desses lugares.

Quanto à aplicação em si, poderá sempre – deverá sempre – ser enriquecida, nomeadamente com conteúdos teóricos que possam anexar-se à componente prática dos percursos traçados, isto é, das rotas propostas para as visitas. Esse enriquecimento pode – deve – ser feito através daquilo que, numa primeira instância, motivou os criadores da iClio a avançar para projetos deste tipo: juntar, às rotas, factos histórias e histórias curiosas sobre os pontos de interesse a visitar. Este trabalho deverá, pois, contar com a ajuda de historiadores, para que os conteúdos apresentados sejam fiáveis e fidedignos e para que o turista nunca se sinta defraudado nas suas expectativas: mais do que visitar, mais do que ver, aprender e conhecer os pontos que visita. Com a junção destas duas funcionalidades, o turismo, as Câmaras Municipais e o país só têm a ganhar.

Considerando a frase de Alan Kay que colocámos na página B deste relatório, que refere que "a melhor maneira de prever o futuro é inventá-lo, acreditamos ter contribuído para uma melhoria do futuro nas visitas a locais abrangidos por plataformas deste tipo e consideramos que, transpostas as barreiras dos problemas, fica a sensação de termos conseguido algo inovador, útil e, consequentemente, muito positivo. Algo que descrevemos neste relatório. Algo que deu "muito gozo" fazer. Algo que esperamos que possa ser útil. Algo que muitos apreciem…

A aplicação por nós criada e descrita neste relatório pode ser usada por diferentes entidades/instituições, nomeadamente relacionadas com o turismo.

Com efeito, como referimos no início deste trabalho, cada vez mais viajamos e, simultaneamente, cada vez mais gostamos de sistemas eficientes para a realização das viagens e, consequentemente, das visitas que pretendem realizar. Não só procuramos viagens rápidas e economicamente acessíveis, mas também viagens que permitam uma boa gestão do tempo, pelo que uma aplicação que permite estabelecer rotas e prever o tempo a dispensar para cada visita pode ser muito útil a milhões de viajantes (nacionais os estrangeiros).

Quando visitamos, de forma autónoma, uma qualquer cidade, por exemplo, normalmente já temos uma noção muito clara do que queremos conhecer, mas nem sempre sabemos como gerir o tempo (muitas vezes curto) que temos para as visitas que queremos efetuar. Daí que, se o Turismo dessa cidade ou a Câmara Municipal, por exemplo, investir nesta aplicação (sempre sujeita a melhoramentos, como é evidente), pode deixar os seus visitantes mais satisfeitos com a visita. E todos sabemos que, turistas (repetimos, nacionais ou estrangeiros) satisfeitos fazem a melhor publicidade que alguma vez existiu: a dos comentários pessoais, considerados fidedignos por serem baseados na experiência pessoal. Para qualquer cidade que aposte no turismo (e, considerando que, como tantas vezes ouvimos, "Portugal está na moda", todas o fazem cada vez mais), esta aplicação será, seguramente, uma mais-valia a considerar.

8.1 Implementações Futuras

Cumpre referir que, se é verdade que há, cada vez mais, aplicações deste género, também é verdade que a aqui descrita contém uma grande dose de inovação, reconhecida por todos quantos aqueles que, até este momento, puderam acompanhar o seu desenvolvimento. Desta forma, trata-se de uma aplicação que, a ser desenvolvida em termos comerciais, poderá trazer grandes vantagens a quem nela apostar e a quem a utilizar.

Melhorias ao sistema como a implementação em larga escala desta ferramenta para um total controlo do fluxo de turistas ou a criação de outros modelos de suporte a esta aplicação através dos registos históricos obtidos pelas mesmas (por exemplo, um ponto de interesse pode ter tendência a ter mais lotação durante um fim de semana ou, por exemplo, entre as 14:00 e as 16:00h em que os bilhetes seriam mais baratos), permitindo assim uma otimização ainda maior do tempo do utilizador.

No entanto, a principal melhoria para implementar neste projeto seria um estudo de dados correto, com um maior número de amostras e com diversas iterações no estudo, para obter os valores parâmetros ideias para os modelos para o caso de estudo, tendo em conta que o estudo realizado a estes dados foi limitado pelos recursos físicos (apenas um único computador que, em cada iteração do estudo dos dados, ficaria inutilizável para qualquer outra tarefa durante 72h) e do custo de tempo associado a este estudo.

Referências

https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/: Rede Neuronal Explicada (consultado em 2018/01/04)

http://pt.euronews.com/2017/11/13/china-cria-sistema-de-vigilancia-capaz-de-reconhecer-caras (consultado em 2018/07/31)

https://observador.pt/2017/12/28/bem-vindos-ao-big-brother-de-orwell-ha-170-milhoes-de-camaras-de-videovigilancia-na-china-e-podem-vir-a-ser-bem-mais/, (consultado em 2018/07/31)

https://pt.wikipedia.org/wiki/Yelp (consultado em 2018/07/31)

https://pt.wikipedia.org/wiki/TripAdvisor (consultado em 2018/07/31)

https://pt.wikipedia.org/wiki/Kayak.com (consultado em 2018/07/31)

https://paginas.fe.up.pt/~mgi99021/it/tipos.htm - Aprendizagem automática(consultado em 2018/07/31)consultado em 2018/07/31)

https://tableless.com.br/o-que-nodejs-primeiros-passos-com-node-js/ (consultado em 2018/08/31)

https://pt.wikipedia.org/wiki/Front-end_e_back-end (consultado em 2018/09/03)

https://www.oficinadanet.com.br/post/13541-afinal-o-que-e-frontend-e-o-que-e-backend- (consultado em 2018/09/03)

https://www.dinheirovivo.pt/fazedores/iclio-somos-so-mais-uma-empresa-de-coimbra-para-o-mundo/ (consultado em 2018/09/03)

https://www.dinheirovivo.pt/fazedores/esta-app-guia-turistas-pelo-mundo/ - iClio (consultado em 2018/09/03)

Anexos

Anexo 1

```python
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import Column, String, Integer, Boolean
from sqlalchemy.orm import relationship, backref

from models.base import Base

class category(Base):
    __tablename__ = 'category'

    id = Column(Integer, primary_key=True)
    CategoryName = Column(String(80))
    CategoryIconName = Column(String(80))
    active = Column(Boolean, default=True)


    def __init__(self, CategoryName, CategoryIconName):
        self.CategoryName = CategoryName
        self.CategoryIconName = CategoryIconName
        self.active = True

    def __repr__(self):
        return '<Category %r' % self.CategoryName
```

Anexo 2

```python
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import Column, String, Integer, ForeignKey, Boolean, FLOAT, JSON
from sqlalchemy.orm import relationship, backref

from models.base import Base

class city(Base):
    __tablename__ = 'city'

    id = Column(Integer, primary_key=True)
    Country = Column(String(80))
    City = Column(String(80))
    geoLatCenter = Column(FLOAT(50))
    geoLongCenter = Column(FLOAT(50))
    distanceMatrix = Column(String(1000000), nullable=False)
    active = Column(Boolean, default=True)


    def __init__(self, City, Country, geoLatCenter, geoLongCenter):
        self.City = City
        self.Country = Country
        self.geoLatCenter = geoLatCenter
        self.geoLongCenter = geoLongCenter
        self.distanceMatrix = '{ "1-1": 0}'
        self.active = True

    def __repr__(self):
        return '<City %r , Country %r>' % self.City, self.Country
```

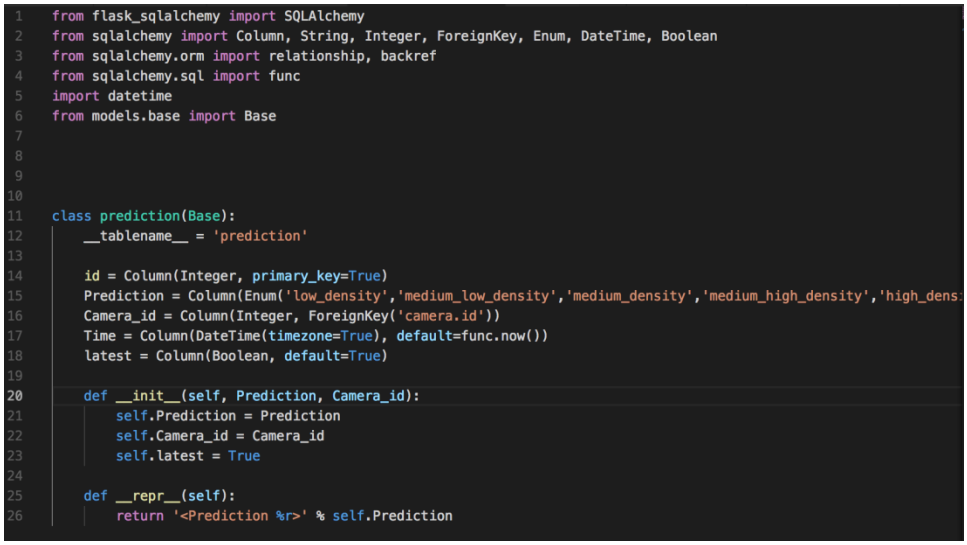
Anexo 3

```python
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import Column, String, Integer, Boolean, FLOAT, ForeignKey
from models.base import Base

class poi(Base):
    __tablename__ = 'poi'

    id = Column(Integer, primary_key=True)
    POIName = Column(String(80))
    Description = Column(String(1000))
    geoLat = Column(FLOAT(50))
    geoLong = Column(FLOAT(50))
    AvgVisitTime = Column(Integer)
    POIArea = Column(FLOAT(50))
    priority = Column(Integer)
    active = Column(Boolean, default=True)
    Category_id = Column(Integer, ForeignKey('category.id'))
    City_id = Column(Integer, ForeignKey('city.id'))

    def __init__(self, POIName, Description, geoLat, geoLong, AvgVisitTime, POIArea, priority, Category_id, City_
        self.POIName = POIName
        self.Description = Description
        self.geoLat = geoLat
        self.geoLong = geoLong
        self.AvgVisitTime = AvgVisitTime
        self.POIArea = POIArea
        self.priority = priority
        self.Category_id = Category_id
        self.City_id = City_id
        self.active = True

    def __repr__(self):
        return '<POI %r>' % self.POIName
```

Anexo 4

```python
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import Column, String, Integer, ForeignKey, Boolean
from sqlalchemy.orm import relationship, backref

from models.base import Base

class camera(Base):
    __tablename__ = 'camera'

    id = Column(Integer, primary_key=True)
    IPaddress = Column(String(80))
    active = Column(Boolean, default=True)
    POI_id = Column(Integer, ForeignKey('poi.id'))


    def __init__(self, IPaddress, POI_id):
        self.IPaddress = IPaddress
        self.active = True
        self.POI_id = POI_id

    def __repr__(self):
        return '<Camera %r>' % self.IPaddress
```

Anexo 5

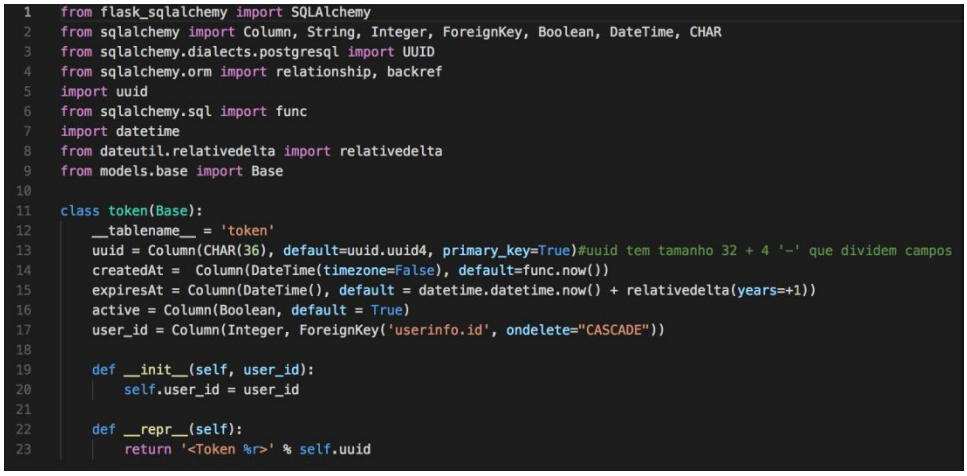


```python
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import Column, String, Integer, ForeignKey, Enum, DateTime, Boolean
from sqlalchemy.orm import relationship, backref
from sqlalchemy.sql import func
import datetime
from models.base import Base




class prediction(Base):
    __tablename__ = 'prediction'

    id = Column(Integer, primary_key=True)
    Prediction = Column(Enum('low_density','medium_low_density','medium_density','medium_high_density','high_dens
    Camera_id = Column(Integer, ForeignKey('camera.id'))
    Time = Column(DateTime(timezone=True), default=func.now())
    latest = Column(Boolean, default=True)

    def __init__(self, Prediction, Camera_id):
        self.Prediction = Prediction
        self.Camera_id = Camera_id
        self.latest = True

    def __repr__(self):
        return '<Prediction %r>' % self.Prediction
```

Anexo 6

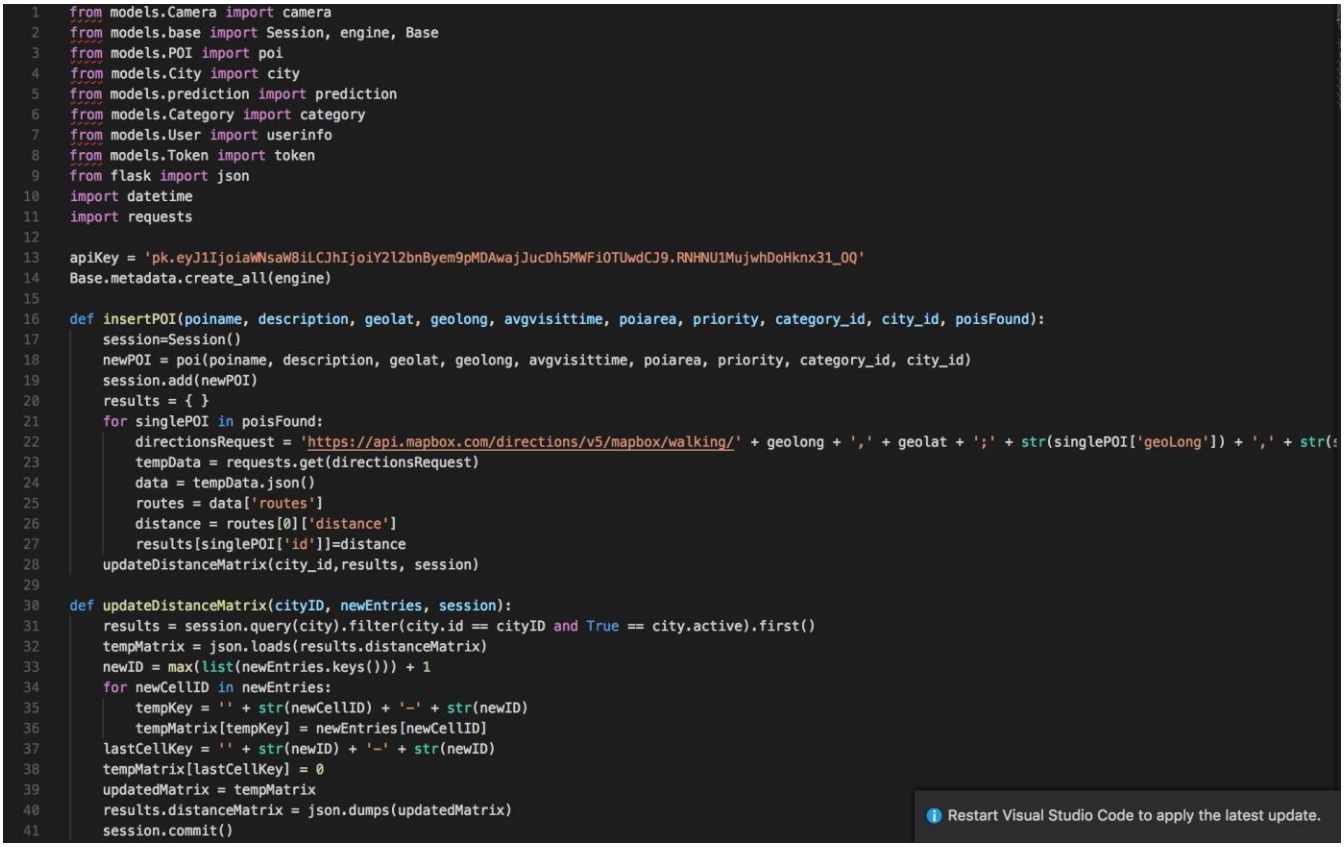
```python
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import Column, String, Integer, ForeignKey, Boolean, CHAR
from sqlalchemy.orm import relationship, backref
import bcrypt
from models.base import Base

class userinfo(Base):
    __tablename__ = 'userinfo'

    id = Column(Integer, primary_key=True)
    email = Column(String(80), unique=True)
    password = Column(String(200))
    role = Column(String(20))

    def __init__(self, email, password, role):
        self.email = email
        # bcrypt.gensalt()))
        # this is a randomly generated salt!
        self.password = str(bcrypt.hashpw(password.encode('utf-8'), b'$2b$12$58be7nNWKicd6NYqvc5sUu'))
        self.role = role
    def __repr__(self):
        return '<User %r>' % self.email
```

Anexo 7



```python
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import Column, String, Integer, ForeignKey, Boolean, DateTime, CHAR
from sqlalchemy.dialects.postgresql import UUID
from sqlalchemy.orm import relationship, backref
import uuid
from sqlalchemy.sql import func
import datetime
from dateutil.relativedelta import relativedelta
from models.base import Base

class token(Base):
    __tablename__ = 'token'
    uuid = Column(CHAR(36), default=uuid.uuid4, primary_key=True)#uuid tem tamanho 32 + 4 '-' que dividem campos
    createdAt =  Column(DateTime(timezone=False), default=func.now())
    expiresAt = Column(DateTime(), default = datetime.datetime.now() + relativedelta(years=+1))
    active = Column(Boolean, default = True)
    user_id = Column(Integer, ForeignKey('userinfo.id', ondelete="CASCADE"))

    def __init__(self, user_id):
        self.user_id = user_id

    def __repr__(self):
        return '<Token %r>' % self.uuid
```

Anexo 8a



```python
from models.Camera import camera
from models.base import Session, engine, Base
from models.POI import poi
from models.City import city
from models.prediction import prediction
from models.Category import category
from models.User import userinfo
from models.Token import token
from flask import json
import datetime
import requests

apiKey = 'pk.eyJ1IjoiaWNsaW8iLCJhIjoiY2l2bnByem9pMDAwajJucDh5MWFiOTUwdCJ9.RNHNU1MujwhDoHknx31_OQ'
Base.metadata.create_all(engine)

def insertPOI(poiname, description, geolat, geolong, avgvisittime, poiarea, priority, category_id, city_id, poisFound):
    session=Session()
    newPOI = poi(poiname, description, geolat, geolong, avgvisittime, poiarea, priority, category_id, city_id)
    session.add(newPOI)
    results = { }
    for singlePOI in poisFound:
        directionsRequest = 'https://api.mapbox.com/directions/v5/mapbox/walking/' + geolong + ',' + geolat + ';' + str(singlePOI['geoLong']) + ',' + str(s
        tempData = requests.get(directionsRequest)
        data = tempData.json()
        routes = data['routes']
        distance = routes[0]['distance']
        results[singlePOI['id']]=distance
    updateDistanceMatrix(city_id,results, session)

def updateDistanceMatrix(cityID, newEntries, session):
    results = session.query(city).filter(city.id == cityID and True == city.active).first()
    tempMatrix = json.loads(results.distanceMatrix)
    newID = max(list(newEntries.keys())) + 1
    for newCellID in newEntries:
        tempKey = '' + str(newCellID) + '-' + str(newID)
        tempMatrix[tempKey] = newEntries[newCellID]
    lastCellKey = '' + str(newID) + '-' + str(newID)
    tempMatrix[lastCellKey] = 0
    updatedMatrix = tempMatrix
    results.distanceMatrix = json.dumps(updatedMatrix)
    session.commit()
```

Restart Visual Studio Code to apply the latest update.

Anexo 8b

```
44  def insertCamera(ip_address,poi_id):
45      session=Session()
46      newCamera = camera(ip_address,poi_id)
47      session.add(newCamera)
48      session.commit()
49
50  def insertPrediction(predict,id_camera):
51      session=Session()
52      newPrediction = prediction(predict,id_camera)
53      session.add(newPrediction)
54      session.commit()
55
56  def insertCity(City,Country,geoLatCenter,geoLongCenter):
57      session=Session()
58      newCity = city(City,Country,geoLatCenter, geoLongCenter)
59      session.add(newCity)
60      session.commit()
61
62  def insertCategory(CategoryName, CategoryIconName):
63      session=Session()
64      newCategory = category(CategoryName, CategoryIconName)
65      session.add(newCategory)
66      session.commit()
67
68  def insertUserInfo(email, password, role):
69      session = Session()
70      newUser = userinfo(email,password,role)
71      session.add(newUser)
72      session.commit()
73
74  def insertToken(user_id):
75      session=Session()
76      newToken = token(user_id)
77      session.add(newToken)
78      session.commit()
79
```

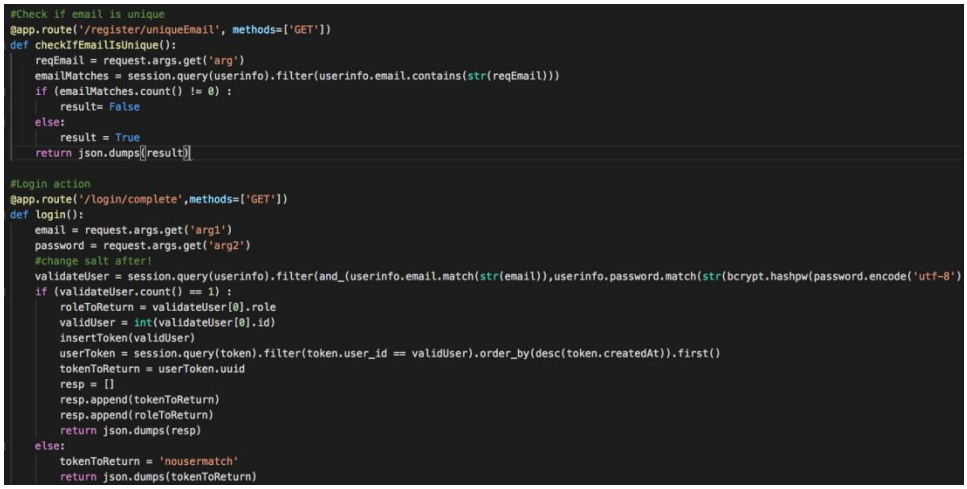
Anexo 9a

```
Base.metadata.create_all(engine)

session = Session()
admin = userinfo("admin@admin.com", "admin", "admin")
user1 = userinfo("testuser1@gmail.com","test1", "user")
user2 = userinfo("testuser2@gmail.com","test2", "user")
user3 = userinfo("testuser3@gmail.com","test3", "user")

session.add(admin)
session.add(user1)
session.add(user2)
session.add(user3)

session.commit()

session = Session()

adminToken = token(1)
user1Token = token(2)
user2Token = token(3)

session.add(adminToken)
session.add(user1Token)
session.add(user2Token)

session.commit()

session= Session()
city1 = city("Coimbra","Portugal",40.223,-8.422)

session.add(city1)
session.commit()
```

Anexo 9b

```
session = Session()
Category1 = category("Attraction","attraction")
Category2 = category("Bus","bus")
Category3 = category("Castle","castle")
Category4 = category("Hospital","hospital")
Category5 = category("Information","information")
Category6 = category("Library","library")
Category7 = category("Lodging","lodging")
Category8 = category("Monument","monument")
Category9 = category("Museum","museum")
Category10 = category("Music","music")
Category11 = category("Park","park")
Category12 = category("Place of Worship","place-of-worship")
Category13 = category("Police","police")
Category14 = category("Rail","rail")
Category15 = category("Stadium","stadium")
Category16 = category("Theatre","theatre")
Category17 = category("Town Hall","town-hall")
Category18 = category("Zoo","zoo")
session.add(Category1)
session.add(Category2)
session.add(Category3)
session.add(Category4)
session.add(Category5)
session.add(Category6)
session.add(Category7)
session.add(Category8)
session.add(Category9)
session.add(Category10)
session.add(Category11)
session.add(Category12)
session.add(Category13)
session.add(Category14)
session.add(Category15)
session.add(Category16)
session.add(Category17)
session.add(Category18)
session.commit()

session = Session()
poi1 = poi("Largo D. Dinis", "", 40.207997, -8.422982, 20, 250, 4, 8, 1)
session.add(poi1)
session.commit()
```

Anexo 10a



```python
#Check if email is unique
@app.route('/register/uniqueEmail', methods=['GET'])
def checkIfEmailIsUnique():
    reqEmail = request.args.get('arg')
    emailMatches = session.query(userinfo).filter(userinfo.email.contains(str(reqEmail)))
    if (emailMatches.count() != 0) :
        result= False
    else:
        result = True
    return json.dumps([result])

#Login action
@app.route('/login/complete',methods=['GET'])
def login():
    email = request.args.get('arg1')
    password = request.args.get('arg2')
    #change salt after!
    validateUser = session.query(userinfo).filter(and_(userinfo.email.match(str(email)),userinfo.password.match(str(bcrypt.hashpw(password.encode('utf-8')
    if (validateUser.count() == 1) :
        roleToReturn = validateUser[0].role
        validUser = int(validateUser[0].id)
        insertToken(validUser)
        userToken = session.query(token).filter(token.user_id == validUser).order_by(desc(token.createdAt)).first()
        tokenToReturn = userToken.uuid
        resp = []
        resp.append(tokenToReturn)
        resp.append(roleToReturn)
        return json.dumps(resp)
    else:
        tokenToReturn = 'nousermatch'
        return json.dumps(tokenToReturn)
```

Anexo 10b

```python
#poi - all data
@app.route('/pois', methods =['GET'])
def get_pois():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        allPOIs = session.query(poi).filter(True == poi.active)
        result=poi_schema.dump(allPOIs)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#camera - all data
@app.route('/cameras', methods =['GET'])
def get_cameras():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        allCameras = session.query(camera).filter(True == camera.active)
        result=camera_schema.dump(allCameras)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#prediction - all data
@app.route('/predictions', methods =['GET'])
def get_predictions():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        allPredictions = session.query(prediction).filter(True == prediction.latest)
        result=prediction_schema.dump(allPredictions)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#city - all data
@app.route('/cities', methods = ['GET'])
def get_cities():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        allCities = session.query(city). filter(True == city.active)
        result = city_schema.dump(allCities)
        return jsonify(result.data)
    else:
        return 'InvalidToken'
```

Anexo 10c

```python
#category - all data
@app.route('/pois/categories', methods = ['GET'])
def get_categories():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        allCategories = session.query(category).filter(True == category.active)
        result = category_schema.dump(allCategories)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#users - all data
@app.route('/users', methods = ['GET'])
def get_users():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        allUsers = session.query(userinfo).all()
        result = userinfo_schema.dump(allUsers)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#poi - findByID
@app.route('/pois/description', methods =['GET'])
def get_poi_by_id():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        poi_id = request.args.get('arg')
        singlePOI = session.query(poi).filter(poi_id == poi.id and True==poi.active)
        result=poi_schema.dump(singlePOI)
        return jsonify(result.data)
    else:
        return 'InvalidToken'
```

Anexo 10d

```python
#camera - findByID
@app.route('/cameras/description', methods =['GET'])
def get_camera_by_id():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        camera_id = request.args.get('arg')
        singleCamera = session.query(camera).filter(camera_id == camera.id and True==camera.active)
        result=camera_schema.dump(singleCamera)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#prediction - findByID
@app.route('/predictions/description', methods =['GET'])
def get_prediction_by_id():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        prediction_id = request.args.get('arg')
        singlePrediction = session.query(prediction).filter(prediction_id == prediction.id and True==prediction.latest)
        result=prediction_schema.dump(singlePrediction)
        return jsonify(result.data)
    else:
        return 'InvalidToken'
#city - findByID
@app.route('/cities/description', methods =['GET'])
def get_city_by_id():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        city_id = request.args.get('arg')
        singleCity = session.query(city).filter(city_id == city.id and True==city.active)
        result=city_schema.dump(singleCity)
        return jsonify(result.data)
    else:
        return 'InvalidToken'
```

Anexo 10e

```python
#users - findByID
@app.route('/users/description', methods = ['GET'])
def get_user_by_id():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        user_id = request.args.get('arg')
        singleUser = session.query(userinfo).filter(user_id == userinfo.id)
        result = userinfo_schema.dump(singleUser)
        return jsonify(result.data)
    else:
        return 'InvalidToken'
#Create
#pois - add poi
@app.route('/pois/add', methods =['POST'])
def post_poi():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        req_data= request.get_json()
        POIName = req_data['POIName']
        Description = req_data['Description']
        geoLat =  req_data['geoLat']
        geoLong = req_data['geoLong']
        AvgVisitTime = req_data['AvgVisitTime']
        POIArea = req_data['POIArea']
        priority =req_data['priority']
        Category_id = req_data['Category_id']
        City_id = req_data['City_id']
        poisFound = req_data['poisFound']
        insertPOI(POIName, Description, geoLat, geoLong, AvgVisitTime, POIArea, priority, Category_id, City_id, poisFound)
        return 'Success'
    else:
        return 'InvalidToken'
```

Anexo 10f

```python
#Find Camera by IP
@app.route('/cameras/findbyip', methods=['GET'])
def findCameraByIP():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        ip = request.args.get('arg')
        allCameras = session.query(camera).filter((camera.IPaddress.contains(str(ip))) & (camera.active==True))
        result=camera_schema.dump(allCameras)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#Find POI by name
@app.route('/pois/findbyname', methods=['GET'])
def findPOIByName():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        name = request.args.get('arg')
        allPOIs = session.query(poi).filter((poi.POIName.contains(str(name))) & (poi.active==True))
        result=poi_schema.dump(allPOIs)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#Find POI by name
@app.route('/pois/findbycity', methods=['GET'])
def findByCity():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        city_id = request.args.get('arg')
        allPOIs = session.query(poi).filter((poi.City_id==city_id) & (poi.active==True))
        result=poi_schema.dump(allPOIs)
        return jsonify(result.data)
    else:
        return 'InvalidToken'
```

Anexo 10g

```python
#Find user by name
@app.route('/users/findbyname', methods=['GET'])
def findUserByName():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        name = request.args.get('arg')
        allUsers = session.query(userinfo).filter(userinfo.email.contains(str(name)))
        result=userinfo_schema.dump(allUsers)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#Find City By name
@app.route('/cities/findbyname', methods=['GET'])
def findCityByName():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        name = request.args.get('arg')
        allCities = session.query(city).filter((city.City.contains(str(name))) & (city.active==True))
        result=city_schema.dump(allCities)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#Find Category By Name
@app.route('/pois/categories/findbyname', methods=['GET'])
def findCategoryByName():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        name = request.args.get('arg')
        allCategories = session.query(category).filter((category.CategoryName.contains(str(name))) & (category.active==True))
        result = category_schema.dump(allCategories)
        return jsonify(result.data)
    else:
        return 'InvalidToken'
```

Anexo 10h

```python
#Find POIs by Category
@app.route('/pois/filterby/categories', methods=['GET'])
def filterPOIsByCategory():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        category_id = request.args.get('arg')
        allPOIs = session.query(poi).filter((poi.Category_id == category_id) & ('True'==poi.active))
        result = poi_schema.dump(allPOIs)
        print(result.data)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#Find POIs by City
@app.route('/pois/filterby/cities', methods=['GET'])
def filterPOIsByCity():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        city_id = request.args.get('arg')
        allPOIs = session.query(poi).filter(poi.City_id == city_id and True==poi.active).order_by(asc(poi.id))
        result = poi_schema.dump(allPOIs)
        return jsonify(result.data)
    else:
        return 'InvalidToken'

#Get distance matrix from city id
@app.route('/cities/distanceMatrix', methods=['GET'])
def getDistanceMatrix():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        city_id = request.args.get('arg')
        singleCity = session.query(city).filter((city.id == city_id) & (city.active==True))
        result = city_schema.dump(singleCity)
        return json.dumps(result.data[0]['distanceMatrix'])
    else:
        return 'InvalidToken'
```

Anexo 11a

```python
#pois - add poi
@app.route('/pois/add', methods =['POST'])
def post_poi():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        req_data= request.get_json()
        POIName = req_data['POIName']
        Description = req_data['Description']
        geoLat =  req_data['geoLat']
        geoLong = req_data['geoLong']
        AvgVisitTime = req_data['AvgVisitTime']
        POIArea = req_data['POIArea']
        priority =req_data['priority']
        Category_id = req_data['Category_id']
        City_id = req_data['City_id']
        poisFound = req_data['poisFound']
        insertPOI(POIName, Description, geoLat, geoLong, AvgVisitTime, POIArea, priority, Category_id, City_id, poisFound)
        return 'Success'
    else:
        return 'InvalidToken'

#cameras - add camera
@app.route('/cameras/add', methods =['POST'])
def post_camera():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        req_data= request.get_json()
        ip_address = req_data['IPaddress']
        poi_id =  req_data['POI_id']
        insertCamera(ip_address, poi_id)
        return 'Success'
    else:
        return 'InvalidToken'
```

Anexo 11b

```python
#predictions - add prediction
@app.route('/predictions/add', methods =['POST'])
def post_prediction():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        req_data= request.get_json()
        predictToAdd = req_data['Prediction']
        IPaddressToAdd =  req_data['IPaddress']
        findByIP = session.query(camera).filter(camera.IPaddress==IPaddressToAdd and True==camera.active).first()
        camera_id = findByIP.id
        findByID = session.query(prediction).filter(prediction.Camera_id == camera_id and True == prediction.latest)
        if(findByID.count() != 0):
            for singleIDFound in findByID:
                singleIDFound.latest=False
        insertPrediction(predictToAdd, camera_id)
        session.commit()
        return 'Success'
    else:
        return 'InvalidToken'

#cities - add city
@app.route('/cities/add', methods = ['POST'])
def post_city():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        req_data = request.get_json()
        cityToAdd = req_data['City']
        CountryToAdd = req_data['Country']
        geoLatCenterToAdd = req_data['geoLatCenter']
        geoLongCenterToAdd = req_data['geoLongCenter']
        insertCity(cityToAdd,CountryToAdd, geoLatCenterToAdd, geoLongCenterToAdd)
        return 'Success'
    else:
        return 'InvalidToken'
```

Anexo 11c

```python
#categories - add category
@app.route('/pois/categories/add', methods = ['POST'])
def post_category():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        req_data = request.get_json()
        categoryToAdd = req_data['CategoryName']
        categoryIconToAdd = req_data['CategoryIconName']
        insertCategory(categoryToAdd, categoryIconToAdd)
        return 'Success'
    else:
        return 'InvalidToken'
```

Anexo 11d

```python
#Get a POI's density
@app.route('/pois/crowd_density', methods=['POST'])
def getCrowdDensity():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token) == True):
        req_data = request.get_json()
        reqPoiList = req_data['poiList']
        finalPredictions = {}
        for singlePOI in reqPoiList:
            temp = singlePOI['id']
            camerasInPOI = session.query(camera).filter((camera.active == True) & (camera.POI_id == temp))
            for singleCamera in camerasInPOI:
                predictions = session.query(prediction).filter((prediction.latest == True) & (prediction.Camera_id == singleCamera.id))
                predictionresults = []
                for singleprediction in predictions:
                    predictionresults.append(singleprediction.Prediction)
            predictionForSingleCamera = max(set(predictionresults), key=predictionresults.count)
            finalPredictions[temp]=predictionForSingleCamera
        print(finalPredictions)
        return json.dumps(finalPredictions)
    else:
        return 'InvalidToken'
```

Anexo 12a

```python
#UPDATE
#pois - update poi
@app.route('/pois/update', methods=['PUT'])
def update_poi():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        poi_id = request.args.get('arg')
        req_data= request.get_json()
        reqPOIName = req_data['POIName']
        reqDescription = req_data['Description']
        reqgeoLat =  req_data['geoLat']
        reqgeoLong = req_data['geoLong']
        reqAvgVisitTime = req_data['AvgVisitTime']
        reqPOIArea = req_data['POIArea']
        reqpriority = req_data['priority']
        reqCategory_id = req_data['Category_id']
        reqCity_id = req_data['City_id']
        findByID = session.query(poi).filter(poi.id==poi_id and True==poi.active).first()
        findByID.POIName=reqPOIName
        findByID.Description = reqDescription
        findByID.geoLat= reqgeoLat
        findByID.geoLong= reqgeoLong
        findByID.AvgVisitTime = reqAvgVisitTime
        findByID.POIArea = reqPOIArea
        findByID.priority = reqpriority
        findByID.Category_id= reqCategory_id
        findByID.City_id= reqCity_id
        session.commit()
        return 'Success'
    else:
        return 'InvalidToken'
```

Anexo 12b

```python
#cameras - update camera
@app.route('/cameras/update', methods=['PUT'])
def update_camera():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        camera_id = request.args.get('arg')
        req_data = request.get_json()
        reqIPaddress = req_data['IPaddress']
        reqPOIid = req_data['POI_id']
        findByID = session.query(camera).filter(camera.id==camera_id and True==camera.active).first()
        findByID.IPaddress =reqIPaddress
        findByID.POI_id = reqPOIid
        session.commit()
        return 'success'
    else:
        return 'InvalidToken'

#predictions - update prediction
@app.route('/predictions/update', methods=['PUT'])
def update_prediction():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        prediction_id = request.args.get('arg')
        req_data = request.get_json()
        reqCameraID= req_data['Camera_id']
        reqPrediction = req_data['Prediction']
        findByID = session.query(prediction).filter(prediction.id == prediction_id and True==prediction.latest).first()
        timestamp = datetime.datetime.now(datetime.timezone.utc)
        findByID.Camera_id = reqCameraID
        findByID.Prediction = reqPrediction
        findByID.Time = timestamp
        session.commit()
        return 'success'
    else:
        return 'InvalidToken'
```

Anexo 12c

```python
#cities - update city
@app.route('/cities/update', methods=['PUT'])
def update_city():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        city_id = request.args.get('arg')
        req_data = request.get_json()
        reqCity = req_data['City']
        reqCountry = req_data['Country']
        reqgeoLatCenter = req_data['geoLatCenter']
        reqgeoLongCenter = req_data['geoLongCenter']
        findByID = session.query(city).filter(city.id == city_id and True == city.active).first()
        findByID.City = reqCity
        findByID.Country = reqCountry
        findByID.geoLatCenter = reqgeoLatCenter
        findByID.geoLongCenter = reqgeoLongCenter
        session.commit()
        return 'success'
    else:
        return 'InvalidToken'

#categories - update category
@app.route('/pois/categories/update', methods=['PUT'])
def update_category():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        category_id = request.args.get('arg')
        req_data = request.get_json()
        reqCategoryName = req_data['CategoryName']
        reqCategoryIconName = req_data['CategoryIconName']
        findByID = session.query(category).filter(category.id == category_id and True == category.active).first()
        findByID.CategoryName = reqCategoryName
        findByID.CategoryIconName = reqCategoryIconName
        session.commit()
        return 'Success'
    else:
        return 'InvalidToken'
```

Anexo 12d

```python
#users - update user role
@app.route('/users/update', methods=['PUT'])
def update_user():
    req_token = request.headers.get('Authorization')
    user_id = request.args.get('arg')
    if(validateToken(req_token)== True and user_id != 1):
        req_data = request.get_json()
        reqRole = req_data['role']
        findByID = session.query(userinfo).filter(userinfo.id == user_id).first()
        findByID.role = reqRole
        session.commit()
        return 'Success'
    else:
        return 'InvalidToken'
```

Anexo 13a

```python
#pois - delete poi by id
@app.route('/pois/delete', methods=['DELETE'])
def del_poi():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        poi_id = request.args.get('arg')
        poiToDelete = session.query(poi).filter(poi.id==poi_id and True==poi.active).first()
        poiToDelete.active = False
        session.commit()
        return 'Success'
    else:
        return 'InvalidToken'

#cameras - delete camera by id
@app.route('/cameras/delete', methods=['DELETE'])
def del_camera():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        camera_id = request.args.get('arg')
        cameraToDelete = session.query(camera).filter(camera.id==camera_id and True == camera.active).first()
        cameraToDelete.active = False
        session.commit()
        return 'Success'
    else:
        return 'InvalidToken'

#predictions - delete prediction by id
@app.route('/predictions/delete', methods=['DELETE'])
def del_prediction():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        prediction_id = request.args.get('arg')
        predictionToDelete = session.query(prediction).filter(prediction.id==prediction_id and True == prediction.latest).first()
        predictionToDelete.latest = False
        session.commit()
        return 'Success'
    else:
        return 'InvalidToken'
```

Anexo 13b

```python
#cities - delete city by id
@app.route('/cities/delete', methods=['DELETE'])
def del_city():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        city_id = request.args.get('arg')
        cityToDelete = session.query(city).filter(city.id == city_id and True == city.active).first()
        cityToDelete.active = False
        session.commit()
        return 'Success'
    else:
        return 'InvalidToken'

#categories - delete category by id
@app.route('/pois/categories/delete', methods=['DELETE'])
def del_category():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        category_id = request.args.get('arg')
        categoryToDelete = session.query(category).filter(category.id == category_id and True == category.active).first()
        categoryToDelete.active = False
        session.commit()
        return 'Success'
    else:
        return 'InvalidToken'

#Users - delete user by id
@app.route('/users/delete', methods=['DELETE'])
def del_user():
    req_token = request.headers.get('Authorization')
    if(validateToken(req_token)== True):
        user_id = request.args.get('arg')
        userToDelete = session.query(userinfo).filter(userinfo.id == user_id).first()
        session.delete(userToDelete)
        session.commit()
        return 'Success'
    else:
        return 'InvalidToken'
```

Anexo 14a

```python
#schemas para conversao de dados para json sem erros
class POISchema(ma.Schema):
    class Meta:
        # Fields to expose
        fields = ('id', 'POIName', 'Description', 'geoLat', 'geoLong', 'AvgVisitTime', 'POIArea', 'priority', 'active', 'Category_id', 'City_id')

poi_schema = POISchema()
poi_schema = POISchema(many=True)

class CameraSchema(ma.Schema):
    class Meta:
        # Fields to expose
        fields = ('id', 'IPaddress', 'POI_id', 'active')

camera_schema = CameraSchema()
camera_schema = CameraSchema(many=True)

class PredictionSchema(ma.Schema):
    class Meta:
        # Fields to expose
        fields = ('id', 'Prediction', 'Camera_id', 'Time', 'latest')

prediction_schema = PredictionSchema()
prediction_schema = PredictionSchema(many=True)

class CitySchema(ma.Schema):
    class Meta:
        #Fields to expose
        fields = ('id', 'City', 'Country', 'geoLatCenter', 'geoLongCenter', 'distanceMatrix', 'active')

city_schema = CitySchema()
city_schema = CitySchema(many=True)

class CategorySchema(ma.Schema):
    class Meta:
        #Fields to expose
        fields = ('id', 'CategoryName', 'CategoryIconName', 'active')

category_schema = CategorySchema()
category_schema = CategorySchema(many=True)
```

Restart Visual Studio Code to apply the latest

Anexo 14b

```python
class UserInfoSchema(ma.Schema):
    class Meta:
        #Fields to expose
        fields = ('id', 'email', 'password', 'role')

userinfo_schema = UserInfoSchema()
userinfo_schema = UserInfoSchema(many=True)

class TokenSchema(ma.Schema):
    class Meta:
        #Fields to expose
        fields = ('uuid', 'createdAt', 'expiresAt', 'active', 'user_id')

token_schema = TokenSchema()
token_schema = TokenSchema(many=True)
```

Anexo 15

```
import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.estimator import regression
from tflearn.metrics import Accuracy
acc = Accuracy()
network = input_data(shape=[None, 28, 28, 1])
# Conv layers
network = conv_2d(network, 64, 3, strides=1, activation='relu', name = 'conv1_3_3_1')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 64, 3, strides=1, activation='relu', name = 'conv1_3_3_2')
network = max_pool_2d(network, 2, strides=2)
# Fully Connected Layer
network = fully_connected(network, 1024, activation='tanh')
# Dropout layer
network = dropout(network, 0.5)
# Fully Connected Layer
network = fully_connected(network, 10, activation='softmax')
# Final network
network = regression(network, optimizer='momentum',
                     loss='categorical_crossentropy',
                     learning_rate=0.001, metric=acc)

# The model with details on where to save
# Will save in current directory
model = tflearn.DNN(network, checkpoint_path='model-', best_checkpoint_path='best-model-')
```

Anexo 16

```
import deepneuralnet as net
import tflearn.datasets.mnist as mnist
# Get the model
model = net.model
# Load data
X, Y, testX, testY = mnist.load_data(one_hot=True)
X = X.reshape([-1, 28, 28, 1])
testX = testX.reshape([-1, 28, 28, 1])
model.fit(X, Y, n_epoch=15, validation_set=(testX, testY), show_metric=True, run_id="deep_nn")
model.save('final-model.tflearn')
```

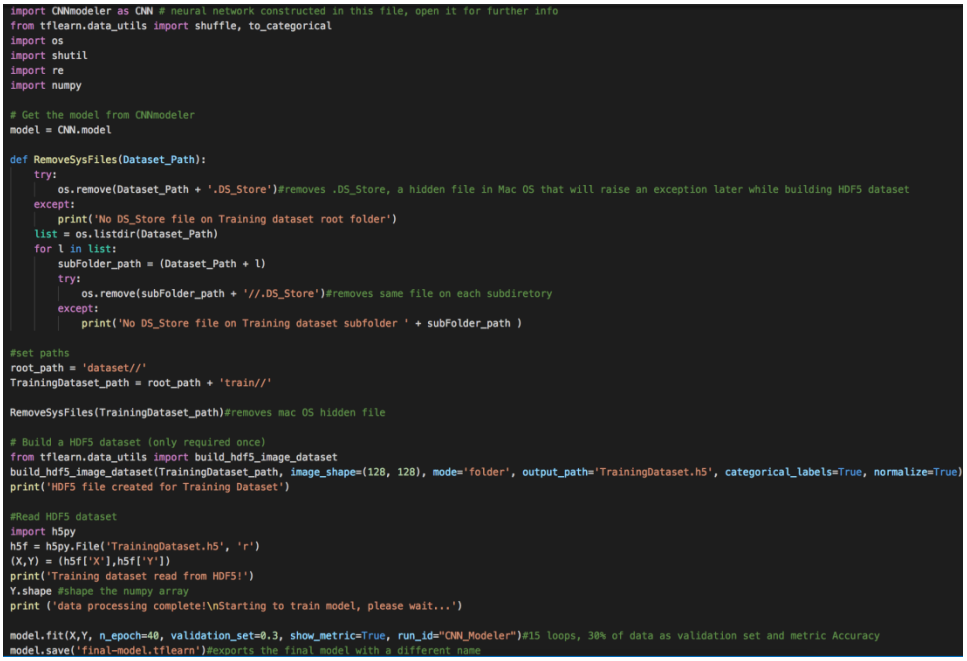
Anexo 17

```
import deepneuralnet as net
import random
import tflearn.datasets.mnist as mnist
from skimage import io
model = net.model
path_to_model = 'final-model.tflearn'
_, _, testX, _ = mnist.load_data(one_hot=True)
model.load(path_to_model)
# Randomly take an image from the test set
rand_index = random.randint(0, len(testX) - 1)
x = testX[rand_index].reshape((28, 28, 1))
result = model.predict([x])[0] # Predict
print (result)
prediction = result.argmax() # The index represents the number predicted in this case
print("Prediction", prediction)
io.imsave('testimage.jpg', x.reshape(28, 28)) # This shows the image in the computer for you to see
```

Anexo 18

```
import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.estimator import regression
from tflearn.metrics import Accuracy
acc = Accuracy()
network = input_data(shape=[None, 128, 128, 3])
# Conv layers
network = conv_2d(network, 64, 3, strides=1, activation='relu', name = 'conv1_3_3_1')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 64, 3, strides=1, activation='relu', name = 'conv1_3_3_2')
network = max_pool_2d(network, 2, strides=2)
# Fully Connected Layer
network = fully_connected(network, 1024, activation='tanh')
# Dropout layer
network = dropout(network, 0.5)
# Fully Connected Layer
network = fully_connected(network, 2, activation='softmax')
# Final network
network = regression(network, optimizer='momentum',
                     loss='categorical_crossentropy',
                     learning_rate=0.01, metric=acc)

# The model and details on which file to export it to
# Will save in current directory
model = tflearn.DNN(network, checkpoint_path='model-', best_checkpoint_path='best-model-')
```

Anexo 19



```
import CNNmodeler as CNN # neural network constructed in this file, open it for further info
from tflearn.data_utils import shuffle, to_categorical
import os
import shutil
import re
import numpy

# Get the model from CNNmodeler
model = CNN.model

def RemoveSysFiles(Dataset_Path):
    try:
        os.remove(Dataset_Path + '.DS_Store')#removes .DS_Store, a hidden file in Mac OS that will raise an exception later while building HDF5 dataset
    except:
        print('No DS_Store file on Training dataset root folder')
    list = os.listdir(Dataset_Path)
    for l in list:
        subFolder_path = (Dataset_Path + l)
        try:
            os.remove(subFolder_path + '//.DS_Store')#removes same file on each subdiretory
        except:
            print('No DS_Store file on Training dataset subfolder ' + subFolder_path )

#set paths
root_path = 'dataset//'
TrainingDataset_path = root_path + 'train//'

RemoveSysFiles(TrainingDataset_path)#removes mac OS hidden file

# Build a HDF5 dataset (only required once)
from tflearn.data_utils import build_hdf5_image_dataset
build_hdf5_image_dataset(TrainingDataset_path, image_shape=(128, 128), mode='folder', output_path='TrainingDataset.h5', categorical_labels=True, normalize=True)
print('HDF5 file created for Training Dataset')

#Read HDF5 dataset
import h5py
h5f = h5py.File('TrainingDataset.h5', 'r')
(X,Y) = (h5f['X'],h5f['Y'])
print('Training dataset read from HDF5!')
Y.shape #shape the numpy array
print ('data processing complete!\nStarting to train model, please wait...')

model.fit(X,Y, n_epoch=40, validation_set=0.3, show_metric=True, run_id="CNN_Modeler")#15 loops, 30% of data as validation set and metric Accuracy
model.save('final-model.tflearn')#exports the final model with a different name
```

Anexo 20

```
import CNNmodeler as CNN
import os
import scipy.misc
import random
from skimage import io

model = CNN.model

def readImage(TestDataset_path):
    dirList = os.listdir(TestDataset_path)
    rand_index = random.randint(0, len(dirList) - 1)
    from PIL import Image, ImageOps
    import numpy as np
    img = Image.open(TestDataset_path + dirList[rand_index]).convert('RGB')
    img.save('test.jpg')
    img = ImageOps.fit(img, ((128,128)), Image.ANTIALIAS)

    img_arr = np.array(img)
    img_arr = img_arr.reshape(128,128,3).astype("float")
    return img_arr

dictionary = {}
dictionary[0]="Cat"
dictionary[1]="Dog"
#set paths
root_path = 'dataset//'
TestDataset_path = root_path + 'test//'

path_to_model = 'final-model.tflearn'
model.load(path_to_model)
img_arr = readImage(TestDataset_path)
prediction = model.predict([img_arr])[0]
print ('Prediction : ' + dictionary[prediction.argmax()])
```

Anexo 21a

```
from __future__ import division
import cv2
import numpy as np
import os

classMap = { }
classMap['low']= 'LOW0'
classMap['medium_low']= 'MEDIUMLOW1'
classMap['medium']= 'MEDIUM2'
classMap['medium_high']= 'MEDIUMHIGH3'
classMap['crowded']= 'CROWDED4'

interval = 10
framesOfAllVideos = []
classList = os.listdir('newDataset/')
ClassFrameCount = { }
getMedianList = []

def median(valList):
    sortedList= sorted(valList)
    listLen = len(valList)
    index = (listLen - 1) // 2
    if (listLen % 2):
        return sortedList[index]
    else:
        return (sortedList[index] + sortedList[index + 1])/2.0

print('Processing videos ')
for eachClass in classList:
    if(eachClass != '.DS_Store'):
        NrFrames = 0
        videoList = os.listdir('newDataset/' + str(eachClass))
        for singleVideo in videoList:
            framesofSingleVideo = []
            framesofSingleVideo.append(eachClass)
```

Anexo 21b

```
            if(singleVideo != '.DS_Store'):
                cap = cv2.VideoCapture('newDataset//' + str(eachClass) + '//' + str(singleVideo))
                fgbg = cv2.createBackgroundSubtractorMOG2()
                while True:
                    ret, frame = cap.read()
                    if frame is None:
                        break
                    if ret:
                        frame = cv2.GaussianBlur(frame, (5,5),0)
                        frame=cv2.GaussianBlur(frame, (5, 5), 0)
                        #apply background subtractor
                        fgmask = fgbg.apply(frame)
                        #filters for the foreground mask
                        cv2.blur(fgmask, (10,10), -1)
                        cv2.threshold(fgmask, 20, 255, cv2.THRESH_BINARY)
                        cv2.medianBlur(fgmask, 3)
                        #removes noise with a 4x4 filter
                        kernel = np.ones((4,4),np.uint8)
                        cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)
                        kernel2 = np.ones((3,3),np.uint8)
                        cv2.morphologyEx(fgmask, cv2.MORPH_ERODE, kernel2)
                        resizedImage = cv2.resize(fgmask,(640,360))
                        framesofSingleVideo.append(resizedImage) #put resized image in list and d
                        NrFrames = NrFrames + 1
                framesOfAllVideos.append(framesofSingleVideo)
        ClassFrameCount[eachClass] = NrFrames
        getMedianList.append(NrFrames)

print('Getting ready to export dataset...')
medianOfFrames = median(getMedianList)
ControlVariable = { }
for key in ClassFrameCount:
    exportFrameNr = medianOfFrames / ClassFrameCount[key]
    ControlVariable[key] = exportFrameNr
```

Anexo 21c

```
print('Getting ready to export dataset...')
medianOfFrames = median(getMedianList)
ControlVariable = { }
for key in ClassFrameCount:
    exportFrameNr = medianOfFrames / ClassFrameCount[key]
    ControlVariable[key] = exportFrameNr

print('Exporting dataset...')
counter = 1
counterXI = 1
for classFrames in framesOfAllVideos:
    classVar = classFrames[0]
    classFrames.pop(0)
    for imageframe in classFrames:
        if(int(interval/ControlVariable[classVar])==0):
            testvar = 1
        else:
            testvar = int(interval/ControlVariable[classVar])
        if(counter % testvar == 0):
            cv2.imwrite('dataset//' + str(classMap[str(classVar)]) + '//' + 'image' + str(counterXI) + '.jpg', imageframe)
            counterXI = counterXI + 1
        counter = counter + 1
        k = cv2.waitKey(30) & 0xff
        if k == 27:
            break
print('Finished successfully!')
```

Anexo 22

```
import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.estimator import regression
from tflearn.metrics import Accuracy
import tensorflow as tf

acc = Accuracy()
network = input_data(shape=[None, 360, 640,1])


# Conv layers
network = conv_2d(network, 64, 3, strides=1, activation='relu', name = 'conv1_3_3_1')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 64, 3, strides=1, activation='relu', name = 'conv1_3_3_2')
network = max_pool_2d(network, 2, strides=2)
# Fully Connected Layer
network = fully_connected(network, 1024, activation='tanh')
# Dropout layer
network = dropout(network, 0.5)
# Fully Connected Layer
network = fully_connected(network, 5, activation='softmax')
# Final network
network = regression(network, optimizer='momentum',
                     loss='categorical_crossentropy',
                     learning_rate=0.01, metric=acc)

# The model and details on which file to export it to
# Will save in current directory
model = tflearn.DNN(network)
```

Anexo 23

Epoch	Tempo	Loss	Accuracy	Validation Loss	Validation Accuracy
1	14268.740s	0.53267	0.7932	0.58430	0.78890
2	13073.059s	0.24115	0.9121	0.00000	0.00000
3	14076.961s	0.17674	0.9421	0.22674	0.91950
4	13154.475s	0.10740	0.9640	0.00000	0.00000
5	14235.503	0.07359	0.9789	0.15895	0.94630
6	15204.317	0.06936	0.9790	0.14321	0.94990
7	14096.677	0.07313	0.9757	0.17395	0.94100
8	14916.568	0.06146	0.9872	0.15716	0.95170
9	14514.666	0.06134	0.9892	0.15774	0.95170
10	14733.750	0.07657	0.9863	0.14933	0.95170
11	13951.030	0.02827	0.9933	0.15063	0.95710
12	14444.682	0.02175	0.9962	0.17085	0.95170
13	14410.371	0.01429	0.9969	0.13124	0.96240
14	14427.140	0.03792	0.9914	0.17947	0.95350
15	14693.741	0.02977	0.9943	0.12173	0.96060

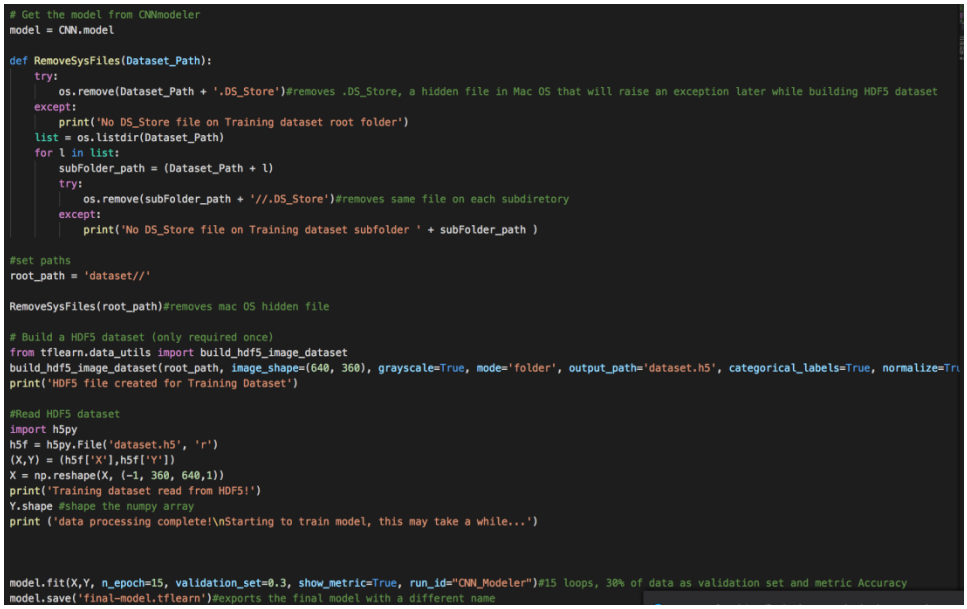
Anexo 24a

```
import cv2
import numpy as np
import os

videoList = os.listdir('tempvideos/')
counterx1 = 0
videocounter=0
for singleVideo in videoList:
    if(singleVideo == '.DS_Store'):
        pass
    else:
        IPaddress = str(singleVideo).replace('.mp4','')
        videocounter=videocounter+1
        valid = False
        cap = cv2.VideoCapture('tempvideos//' + singleVideo)
        first_iter = True
        last300Frames = []
        last300UntouchedFrames = []
        last300WhitePixelCount = []
        valid = False
        fgbg = cv2.createBackgroundSubtractorMOG2()
        width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
        height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
        TotalArea=width*height
        print('loading video number ' + str(videocounter)+ '...')
        while(valid!=True):
            ret, frame = cap.read()
            if first_iter:
                avg = np.float32(frame)
                first_iter = False
            if frame is None:
                print(len(last300Frames))
                break
            if ret:
                cv2.accumulateWeighted(frame, avg, 0.003)
                background = cv2.convertScaleAbs(avg)
                frame=cv2.GaussianBlur(frame, (5, 5), 0)
                #apply background subtractor
                fgmask = fgbg.apply(frame)
                #filters for the foreground mask
                cv2.blur(fgmask, (10,10), -1)
                cv2.threshold(fgmask, 20, 255, cv2.THRESH_BINARY)
                cv2.medianBlur(fgmask, 3)
```

Anexo 24b

```
            #removes noise with a 4x4 filter
            kernel = np.ones((4,4),np.uint8)
            cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)
            kernel2 = np.ones((3,3),np.uint8)
            cv2.morphologyEx(fgmask, cv2.MORPH_ERODE, kernel2)
            last300Frames.append(fgmask)
            if(len(last300Frames)>200):
                last300Frames.pop(0)
                for singleFrame in last300Frames:
                    #count non-black pixels for initial estimate
                    temp1 = cv2.countNonZero(singleFrame)
                    whitearea= temp1/TotalArea
                    last300WhitePixelCount.append(whitearea)
                maxPC = 0
                minPC = TotalArea
                for pixelCount in last300WhitePixelCount:
                    if(pixelCount<minPC):
                        minPC = pixelCount
                    if(pixelCount>maxPC):
                        maxPC = pixelCount
                PCvariation = maxPC-minPC
                if(PCvariation < 0.25):# max percentage of variation
                    valid = True
                else:
                    valid = False
                print('variation: '+ str(PCvariation))
                last300WhitePixelCount = []

    if(valid):
        counterx25 = 0
        for singleFrame in last300Frames:
            resizedImage = cv2.resize(singleFrame,(640,360))
            if(counterx25%25==0):
                if(counterx1 != 0):
                    cv2.imwrite('tempimages/' + IPaddress + '_' + str(counterx1) + '.jpg',resizedImage)
                counterx1 = counterx1 + 1

            k = cv2.waitKey(30) & 0xff
            if k == 27:
                break
            counterx25 = counterx25 + 1
        os.remove('tempvideos//' + singleVideo) disabled for test purposes
```

Anexo 25



```
# Get the model from CNNmodeler
model = CNN.model

def RemoveSysFiles(Dataset_Path):
    try:
        os.remove(Dataset_Path + '.DS_Store')#removes .DS_Store, a hidden file in Mac OS that will raise an exception later while building HDF5 dataset
    except:
        print('No DS_Store file on Training dataset root folder')
    list = os.listdir(Dataset_Path)
    for l in list:
        subFolder_path = (Dataset_Path + l)
        try:
            os.remove(subFolder_path + '//.DS_Store')#removes same file on each subdiretory
        except:
            print('No DS_Store file on Training dataset subfolder ' + subFolder_path )

#set paths
root_path = 'dataset//'

RemoveSysFiles(root_path)#removes mac OS hidden file

# Build a HDF5 dataset (only required once)
from tflearn.data_utils import build_hdf5_image_dataset
build_hdf5_image_dataset(root_path, image_shape=(640, 360), grayscale=True, mode='folder', output_path='dataset.h5', categorical_labels=True, normalize=Tru
print('HDF5 file created for Training Dataset')

#Read HDF5 dataset
import h5py
h5f = h5py.File('dataset.h5', 'r')
(X,Y) = (h5f['X'],h5f['Y'])
X = np.reshape(X, (-1, 360, 640,1))
print('Training dataset read from HDF5!')
Y.shape #shape the numpy array
print ('data processing complete!\nStarting to train model, this may take a while...')


model.fit(X,Y, n_epoch=15, validation_set=0.3, show_metric=True, run_id="CNN_Modeler")#15 loops, 30% of data as validation set and metric Accuracy
model.save('final-model.tflearn')#exports the final model with a different name
```

Anexo 26a

```
<template>
  <div id="app">
    <b-navbar type="dark" variant="primary" toggleable>
    <b-navbar-toggle target="nav_dropdown_collapse"></b-navbar-toggle>
      <b-collapse is-nav id="nav_dropdown_collapse">
        <b-navbar-nav>
          <b-nav-item href="/">Home</b-nav-item>
          <b-nav-item v-if="role==='user'" href="/route">New Route</b-nav-item>
          <b-nav-item-dropdown v-if="role==='admin'" text="List" left>
            <b-dropdown-item href="/poi">poi</b-dropdown-item>
            <b-dropdown-item href="/camera">camera</b-dropdown-item>
            <b-dropdown-item href="/prediction">prediction</b-dropdown-item>
            <b-dropdown-item href="/category">category</b-dropdown-item>
            <b-dropdown-item href="/city">city</b-dropdown-item>
            <b-dropdown-item href="/users">users</b-dropdown-item>
          </b-nav-item-dropdown>
          <b-nav-item v-if="role!=='admin'" href="/about">About</b-nav-item>
        </b-navbar-nav>
        <b-navbar-nav v-if="showLogin" class="ml-auto">
          <b-nav-item href="/login">Login</b-nav-item>
          <b-nav-item href="/register">Register</b-nav-item>
        </b-navbar-nav>
        <b-navbar-nav v-if="!showLogin" class="ml-auto">
          <b-nav-text>{{user}}</b-nav-text>
          <b-button type="dark" variant="primary" @click="logout">Logout</b-button>
        </b-navbar-nav>
      </b-collapse>
    </b-navbar>
    <router-view/>
  </div>
</template>

<script>
import LogoutService from '@/services/LogoutService'
```

Anexo 26b

```
export default {
  name: 'App',
  data () {
    return {
      showLogin: true,
      role: 'visitor',
      token: null,
      user: null
    }
  },
  methods: {
    getPreviousSessionInfo () {
      if (this.$cookies.get('cdcToken') !== null) {
        this.token = this.$cookies.get('cdcToken')
        this.role = this.$cookies.get('cdcRole')
        this.user = this.$cookies.get('cdcUser')
        this.showLogin = false
      } else {
        this.showLogin = true
      }
    },
    async logout () {
      var _this = this
      try {
        await LogoutService.logout({
          token: this.token
        })
          .then(response => {
            _this.$cookies.remove('cdcToken')
            _this.$cookies.remove('cdcRole')
            _this.showLogin = true
            _this.updateNav(null, 'visitor')
          })
      } catch (error) {
        console.log(error)
      }
    },
    // getters for child components
    getToken () {
      return this.token
    },
```

Anexo 26c

```
    getRole () {
      return this.role
    },
    getUser () {
      return this.user
    },
    // updates navigation bar according to role and token
    updateNav (token, role, email) {
      if (token !== null) {
        this.showLogin = false
        this.role = role
        this.token = token
        this.user = email
      } else {
        this.showLogin = true
        this.role = 'visitor'
        this.token = null
        this.user = null
        this.$router.push('/')
      }
    }
  },
  created () {
    this.getPreviousSessionInfo()
  }
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  width: 100%;
  display: flex;
  flex-direction: column;
}
</style>
```

Anexo 27

```
import Vue from 'vue'
import Router from 'vue-router'
const routerOptions = [
  { path: '/', component: 'Home' },
  { path: '/about', component: 'About' },
  { path: '/register', component: 'Register' },
  { path: '/login', component: 'Login' },
  { path: '/poi', component: 'POI' },
  { path: '/poi/add', component: 'AddPOI' },
  { path: '/camera', component: 'Camera' },
  { path: '/camera/add', component: 'AddCamera' },
  { path: '/prediction', component: 'Prediction' },
  { path: '/category', component: 'Category' },
  { path: '/category/add', component: 'AddCategory' },
  { path: '/city', component: 'City' },
  { path: '/city/add', component: 'AddCity' },
  { path: '/users', component: 'User' },
  { path: '/route', component: 'Route' },
  { path: '*', component: 'NotFound' }
]

const routes = routerOptions.map(route => {
  return {
    ...route,
    component: () => import(`@/components/${route.component}.vue`)
  }
})
Vue.use(Router)

export default new Router({
  routes,
  mode: 'history'
})
```

Anexo 28

```
import Api from '@/services/Api'

export default {
  checkIfEmailisUnique (reqdata) {
    return Api().get('/register/uniqueEmail?arg=' + reqdata.email)
  }
}
```

Anexo 29a

```
<template>
  <div>
    <h1>Register</h1>
      <input
      v-model="email"
      placeholder="Email"
      v-on:input="validateEmail()"/>
      <br/>
      <input
      type="password"
      v-model="password"
      placeholder="Password"
      v-on:input="validatePassword()"/>
      <br>
      <b-button :disabled="!validSubmit" @click="submit">Register</b-button>
      <br>
      <div class="error" v-html="error" />
      <div class="success" v-html="success" />
  </div>
</template>
<script>
import CheckIfEmailisUniqueService from '@/services/CheckIfEmailisUniqueService'
import AddUserInfoService from '@/services/AddUserInfoService'
var emailValidator = require('email-validator')
export default {
  data () {
    return {
      email: '',
      password: '',
      validEmail: false,
      validPassword: false,
      uniqueEmail: false,
      validSubmit: false,
      error: null,
      success: null
    }
  },
```

Anexo 29b

```
methods: {
  // validations
  validateEmail () {
    if (emailValidator.validate(this.email) === true) {
      this.validEmail = true
    } else {
      this.validEmail = false
    }
    this.submitValidated()
  },
  validatePassword () {
    if (this.password.length > 4 && this.password.length < 16) {
      this.validPassword = true
    } else {
      this.validPassword = false
    }
    this.submitValidated()
  },
  submitValidated () {
    if (this.validPassword === true && this.validEmail === true) {
      this.validSubmit = true
    } else {
      this.validSubmit = false
    }
  },
  // submit
  async submit () {
    await this.verifyUniqueEmail()
    if (this.uniqueEmail === true) {
      await this.RegisterComplete()
    } else {
      this.success = null
      this.error = 'please choose a valid email'
    }
  },
```

Anexo 29c

```
    // verify if email is unique
    async verifyUniqueEmail () {
      try {
        await CheckIfEmailisUniqueService.checkIfEmailisUnique({
          email: this.email
        }).then(response => {
          this.uniqueEmail = response.data
        })
      } catch (error) {
        console.log(error)
      }
    },
    // add userInfo to DB
    async RegisterComplete () {
      try {
        await AddUserInfoService.addUserInfo({
          email: this.email,
          password: this.password
        }).then(response => {
          this.success = 'register successful'
          this.error = null
          this.$router.push('../login')
        })
      } catch (error) {
        console.log(error)
      }
    }
  }
}
</script>
<style scoped>
.error {
  color: red;
}
.success {
  color: green;
}
</style>
```

Anexo 30

```
import Api from '@/services/Api'

export default {
  login (userinfo) {
    return Api().get('/login/complete?arg1=' + userinfo.email + '&arg2=' + userinfo.password)
  }
}
```

Anexo 31a

```
<template>
  <div>
    <h1>Login</h1>
    <input
    v-model="email"
    placeholder="Email"/>
    <br/>
    <input
    type="password"
    v-model="password"
    placeholder="Password"/>
    <br>
    <b-button @click="submit">Login</b-button>
    <br>
    <div class="error" v-html="error" />
    <div class="success" v-html="success" />
  </div>
</template>
<script>
import LoginService from '@/services/LoginService'
export default {
  data () {
    return {
      email: '',
      password: '',
      error: null,
      success: null
    }
  },
  methods: {
    async submit () {
      try {
        await this.login()
      } catch (error) {
        console.log(error)
      }
    },
```

Anexo 31b

```
    async login () {
      try {
        await LoginService.login({
          email: this.email,
          password: this.password
        }).then(response => {
          if (response.data === 'nousermatch') {
            this.success = null
            this.error = 'please input valid information'
          } else {
            this.error = null
            this.success = 'login successfull'
            this.$cookies.set('cdcToken', response.data[0])
            this.$cookies.set('cdcRole', response.data[1])
            this.$cookies.set('cdcUser', this.email)
            this.$parent.updateNav(response.data[0], response.data[1], this.email)
            this.$router.push('/')
          }
        })
      } catch (error) {
        console.log(error)
      }
    }
  }
}
</script>
<style scoped>
.error {
  color: red;
}
.success {
  color: green;
}
</style>
```

Anexo 32

```
import Api from '@/services/Api'

export default {
  findByName (pois, cookie) {
    return Api().get('/pois/findbyname?arg=' + pois.POIName, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 33

```
import Api from '@/services/Api'

export default {
  updatePOI (poiInfo, poiID, cookie) {
    return Api().put('/pois/update?arg=' + poiID, poiInfo, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 34

```
import Api from '@/services/Api'

export default {
  deletePOI (poiID, cookie) {
    return Api().delete('/pois/delete?arg=' + poiID, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 35

```
import Api from '@/services/Api'

export default {
  findCategoryByName (categories, cookie) {
    return Api().get('/pois/categories/findbyname?arg=' + categories.CategoryName, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 36

```
import Api from '@/services/Api'

export default {
  findCityByName (cities, cookie) {
    return Api().get('/cities/findbyname?arg=' + cities.City, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 37a

```
<template>
  <div>
    <br>
    <h1 v-if="showlist">POIS</h1>
    <label v-if="showlist">Choose POI from the list:
    <input list=results name="searchbar" v-model="searchbar" @focus="findByName" @change="findByName" /></label>
    <b-button v-if="showlist" class="newButton" href="http://localhost:8080/poi/add">New...</b-button>
    <datalist id="results">
      <li v-for="POIFound in pois" v-bind:key="POIFound.id">
        <option>{{POIFound.POIName}}</option>
      </li>
    </datalist>
    <br>
    <br>
    <table align="center" v-if="showlist" >
      <tr>
        <td>ID</td>
        <td>Name</td>
        <td>Description</td>
        <td>Latitude</td>
        <td>Longitude</td>
        <td>Average Visit Time</td>
        <td>Area</td>
        <td>Priority</td>
        <td>Category</td>
        <td>City</td>
      </tr>
      <tr v-for="poi in pois" v-bind:key="poi.id">
        <td>
          {{poi.id}}
        </td>
        <td>
          {{poi.POIName}}
        </td>
        <td>
          {{poi.Description}}
        </td>
        <td>
          {{poi.geoLat}}
        </td>
        <td>
          {{poi.geoLong}}
        </td>
```

Anexo 37b

```
      <td>
        {{poi.AvgVisitTime}}
      </td>
      <td>
        {{poi.POIArea}}
      </td>
      <td>
        {{poi.priority}}
      </td>
      <td v-for="category in categories" v-bind:key="category.id + '-cat'" v-if="poi.Category_id === category.id">
        {{category.CategoryName}}
      </td>
      <td v-for="city in cities" v-bind:key="city.id + '-city'" v-if="poi.City_id === city.id">
        {{city.City}}
      </td>
      <td class="buttons">
        <b-button @click="updateButtonPressed(poi.id)">
          Update
        </b-button>
      </td>
      <td class="buttons">
        <b-button @click="deleteButtonPressed(poi.id)">
          Delete
        </b-button>
      </td>
    </tr>
  </table>
  <h1 v-if="showupdate">Edit POI</h1>
  <input
    type="POIName"
    name="POIName"
    v-if="showupdate"
    v-model="poiToUpdate.POIName"
    :placeholder="poiToUpdate.POIName"  />
    <br>
    <input
    type="Description"
    name="Description"
    v-if="showupdate"
    v-model="poiToUpdate.Description"
    :placeholder="poiToUpdate.Description"  />
    <br>
```

Anexo 37c

```
<input
type="geoLat"
name="geoLat"
v-if="showupdate"
v-model="poiToUpdate.geoLat"
:placeholder="poiToUpdate.geoLat" />
<br>
<input
type="geoLong"
name="geoLong"
v-if="showupdate"
v-model="poiToUpdate.geoLong"
:placeholder="poiToUpdate.geoLong" />
<br>
<input
type="number"
name="AvgVisitTime"
v-if="showupdate"
v-model="poiToUpdate.AvgVisitTime"
:placeholder="poiToUpdate.AvgVisitTime"  />
<br>
<input
type="number"
name="POIArea"
v-if="showupdate"
v-model="poiToUpdate.POIArea"
:placeholder="poiToUpdate.POIArea"  />
<br>
<input
type="number"
name="priority"
v-if="showupdate"
v-model="poiToUpdate.priority"
:placeholder="poiToUpdate.priority"  />
<br>
<input list=categoryresults v-if="showupdate" name="categoryname" v-model="categoryname" @focus="findCategoryByName" @input="findCategoryByName"/>
<datalist id="categoryresults">
  <li v-for="categoryFound in categories" v-bind:key="categoryFound.id + '-cat-Found'">
    <option>{{categoryFound.CategoryName}}</option>
  </li>
</datalist>
```

Anexo 37d

```
      <input list=cityresults v-if="showupdate" name="cityname" v-model="cityname" @focus="findCityByName" @input="findCityByName"/>
      <datalist id="cityresults">
        <li v-for="cityFound in cities" v-bind:key="cityFound.id + '-city-Found'">
          <option>{{cityFound.City}}</option>
        </li>
      </datalist>
      <br>
    <b-button :disabled="!enablebutton" v-if="showupdate" @click="updateConfirmPressed(poiToUpdate.id)">Confirm</b-button>
    <b-button v-if="showupdate" @click="back">Cancel</b-button>
    <div class="error" v-html="error" />
    <div class="success" v-html="success" />
    <div class="nomatches" v-html="nomatches" />
  </div>
</template>
<script>
import axios from 'axios'
import FindByNameService from '@/services/FindByNameService'
import DeletePOIService from '@/services/DeletePOIService'
import UpdatePOIService from '@/services/UpdatePOIService'
import FindCategoryByNameService from '@/services/FindCategoryByNameService'
import FindCityByNameService from '@/services/FindCityByNameService'
export default {
  data () {
    return {
      pois: [],
      poiToUpdate: [],
      cities: [],
      categories: [],
      categoriesForUpdate: '',
      categoryname: '',
      cityname: '',
      searchbar: '',
      success: null,
      error: null,
      result: null,
      nomatches: null,
      showlist: true,
      showupdate: false,
      validCity: false,
      validCategory: false,
      enablebutton: false
    }
  },
```

Anexo 37e

```
methods: {
  back () {
    window.location.reload()
  },
  getPOIs () {
    this.pois = this.getPOIsFromBackend()
  },
  getPOIsFromBackend () {
    const path = `http://localhost:5000/pois`
    axios.get(path, {
      headers: {
        Authorization: `${this.$cookies.get('cdcToken')}`
      }
    })
      .then(response => {
        this.pois = response.data
      })
      .catch(error => {
        console.log(error)
      })
  },
  getCities () {
    this.cities = this.getCitiesFromBackend()
  },
  getCitiesFromBackend () {
    const path = `http://localhost:5000/cities`
    axios.get(path, {
      headers: {
        Authorization: `${this.$cookies.get('cdcToken')}`
      }
    })
      .then(response => {
        this.cities = response.data
      })
      .catch(error => {
        console.log(error)
      })
  },
  getCategories () {
    this.categories = this.getCategoriesFromBackend()
  },
```

Anexo 37f

```
getCategoriesFromBackend () {
  const path = `http://localhost:5000/pois/categories`
  axios.get(path, {
    headers: {
      Authorization: `${this.$cookies.get('cdcToken')}`
    }
  })
    .then(response => {
      this.categories = response.data
    })
    .catch(error => {
      console.log(error)
    })
},
deleteButtonPressed (poiID) {
  var _this = this
  this.$dialog.confirm('Please confirm to continue')
    .then(function () {
      try {
        DeletePOIService.deletePOI(poiID, _this.$parent.getToken())
      } catch (error) {
        console.log(error) // not working
      }
      window.location.reload()
    })
},
```

Anexo 37g

```
async getPOIByIDFromBackend (poiid) {
  const path = `http://localhost:5000/pois/description?arg=` + poiid
  try {
    await axios.get(path, {
      headers: {
        Authorization: this.$parent.getToken()
      }
    })
      .then(response => {
        this.poiToUpdate = response.data[0]
      })
      .catch(error => {
        console.log(error)
      })
  } catch (error) {
    console.log(error)
  }
},
async updateButtonPressed (poiID) {
  this.showlist = false
  this.poiToUpdate = this.getPOIByIDFromBackend(poiID)
  this.showupdate = true
},
async updateConfirmPressed (poiID) {
  try {
    await UpdatePOIService.updatePOI({
      POIName: this.poiToUpdate.POIName,
      Description: this.poiToUpdate.Description,
      geoLat: this.poiToUpdate.geoLat,
      geoLong: this.poiToUpdate.geoLong,
      AvgVisitTime: this.poiToUpdate.AvgVisitTime,
      POIArea: this.poiToUpdate.POIArea,
      priority: this.poiToUpdate.priority,
      Category_id: this.categoriesForUpdate[0].id,
      City_id: this.citiesForUpdate[0].id
    }, poiID, this.$parent.getToken())
    window.location.reload()
  } catch (error) {
    console.log(error)
  }
  this.success = 'Updated Successfully'
},
```

Anexo 37h

```
async findByName () {
  try {
    await FindByNameService.findByName({
      POIName: this.searchbar
    }, this.$parent.getToken()).then(response => {
      this.pois = response.data
    })
    if (this.pois.length !== 0) {
      this.success = 'Found Successfully'
      this.error = null
      this.nomatches = null
    } else {
      this.error = null
      this.success = null
      this.nomatches = 'No matches found'
    }
  } catch (error) {
    this.error = 'Something happened'
    this.success = null
    this.nomatches = null
  }
  this.validateOutput()
},
async findCategoryByName () {
  try {
    await FindCategoryByNameService.findCategoryByName({
      CategoryName: this.categoryname
    }, this.$parent.getToken()).then(response => {
      this.categoriesForUpdate = response.data
    })
    if (this.categoriesForUpdate.length === 1 && this.categoryname === this.categoriesForUpdate[0].CategoryName) {
      this.validCategory = true
    } else {
      this.validCategory = false
    }
  } catch (error) {
    console.log(error)
  }
  this.validateOutput()
},
```

Anexo 37i

```
    validateOutput () {
      if (this.validCity === true && this.validCategory === true) {
        this.enablebutton = true
      } else {
        this.enablebutton = false
      }
    },
    async findCityByName () {
      try {
        await FindCityByNameService.findCityByName({
          City: this.cityname
        }, this.$parent.getToken()).then(response => {
          this.citiesForUpdate = response.data
        })
        if (this.citiesForUpdate.length === 1 && this.cityname === this.citiesForUpdate[0].City) {
          this.validCity = true
        } else {
          this.validCity = false
        }
      } catch (error) {
        console.log(error)
      }
      this.validateOutput()
    }

  },
  created () {
    this.getPOIs()
    this.getCities()
    this.getCategories()
  }
}
```

Anexo 38

```
import Api from '@/services/Api'

export default {
  addPOI (poiInfo, cookie) {
    return Api().post('/pois/add', poiInfo, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 39a

```
<template>
  <div>
  <br>
  <br>
    <h1>Add POI</h1>
    <input
      type="text"
      name="POIName"
      v-model="POIName"
      placeholder="Name" />
      <br>
      <input
      type="text"
      name="Description"
      v-model="Description"
      placeholder="Description" />
      <br>
      <input
      type="geoLat"
      name="geoLat"
      v-model="geoLat"
      placeholder="Latitude"
      v-on:input="geoLatValidated()"/>
      <br>
      <input
      type="geoLong"
      name="geoLong"
      v-model="geoLong"
      placeholder="Longitude"
      v-on:input="geoLongValidated()"/>
      <br>
      <input
      type="number"
      name="AvgVisitTime"
      v-model="AvgVisitTime"
      placeholder="Average Visit Time"  />
      <br>
      <input
      type="number"
      name="POIArea"
      v-model="POIArea"
      placeholder="Area"/>
      <br>
```

Anexo 39b

```
    <input
    type="number"
    name="priority"
    v-model="priority"
    placeholder="Priority (Min:1, Max:5)"
     />
    <br>
    <input list=categoryresults name="categoryname" v-model="categoryname" @focus="findCategoryByName" @input="findCategoryByName" placeholder="Category"/
    <datalist id="categoryresults">
      <li v-for="categoryFound in categories" v-bind:key="categoryFound.id + '-cat-Found'">
        <option>{{categoryFound.CategoryName}}</option>
      </li>
    </datalist>
    <br>
    <input list=cityresults name="cityname" v-model="cityname" @focus="findCityByName" @input="findCityByName" placeholder="City"/>
    <datalist id="cityresults">
      <li v-for="cityFound in cities" v-bind:key="cityFound.id + '-city-Found'">
        <option>{{cityFound.City}}</option>
      </li>
    </datalist>
    <br>
    <div class="error" v-html="error" />
    <div class="success" v-html="success" />
    <br>
    <b-button  :disabled="!validated"
      @click="addPOI">
      Add POI
    </b-button>
    <b-button @click="back">Back</b-button>
  </div>
</template>
```

Anexo 39c

```
<script>
import axios from 'axios'
import AddPOIService from '@/services/AddPOIService'
import FindCategoryByNameService from '@/services/FindCategoryByNameService'
import FindCityByNameService from '@/services/FindCityByNameService'
export default {
  data () {
    return {
      POIName: '',
      Description: '',
      geoLat: '',
      geoLong: '',
      AvgVisitTime: '',
      POIArea: null,
      priority: null,
      cities: [],
      categories: [],
      poisFound: [],
      citiesFound: [],
      categoriesFound: [],
      cityname: '',
      categoryname: '',
      validLat: false,
      validLong: false,
      validCity: false,
      validCategory: false,
      validated: false,
      error: null,
      success: null
    }
  },
```

Anexo 39d

```
methods: {
  async addPOI () {
    try {
      await this.findPOIsByCityID(this.citiesFound[0].id)
      await AddPOIService.addPOI({
        POIName: this.POIName,
        Description: this.Description,
        geoLat: this.geoLat,
        geoLong: this.geoLong,
        AvgVisitTime: this.AvgVisitTime,
        POIArea: this.POIArea,
        priority: this.priority,
        Category_id: this.categoriesFound[0].id,
        City_id: this.citiesFound[0].id,
        poisFound: this.poisFound
      }, this.$parent.getToken())
      this.success = 'Added Successfully'
      this.$router.push('../poi')
    } catch (error) {
      this.error = error.response.data.error
    }
  },
  getCities () {
    this.cities = this.getCitiesFromBackend()
  },
  getCitiesFromBackend () {
    const path = `http://localhost:5000/cities`
    axios.get(path, {
      headers: {
        Authorization: this.$parent.getToken()
      }
    })
      .then(response => {
        this.cities = response.data
      })
      .catch(error => {
        console.log(error)
      })
  },
  getCategories () {
    this.categories = this.getCategoriesFromBackend()
  },
```

Anexo 39e

```
getCategoriesFromBackend () {
  const path = `http://localhost:5000/pois/categories`
  axios.get(path, {
    headers: {
      Authorization: this.$parent.getToken()
    }
  })
    .then(response => {
      this.categories = response.data
    })
    .catch(error => {
      console.log(error)
    })
},
back () {
  this.$router.push('../poi')
},
geoLatValidated () {
  if (this.geoLat > -90 && this.geoLat < 90) {
    this.validLat = true
  } else {
    this.validLat = false
  }
  this.submitvalidated()
},
geoLongValidated () {
  if (this.geoLong > -180 && this.geoLong < 180) {
    this.validLong = true
  } else {
    this.validLong = false
  }
  this.submitvalidated()
},
submitvalidated () {
  if (this.validLat === true && this.validLong === true && this.validCity === true && this.validCategory === true) {
    this.validated = true
  } else {
    this.validated = false
  }
},
```

Anexo 39f

```
async findCityByName () {
  try {
    await FindCityByNameService.findCityByName({
      City: this.cityname
    }, this.$parent.getToken()).then(response => {
      this.citiesFound = response.data
    })
    if (this.citiesFound.length === 1 && this.cityname === this.citiesFound[0].City) {
      this.validCity = true
    } else {
      this.validCity = false
    }
  } catch (error) {
    console.log(error)
  }
  this.submitvalidated()
},
async findCategoryByName () {
  try {
    await FindCategoryByNameService.findCategoryByName({
      CategoryName: this.categoryname
    }, this.$parent.getToken()).then(response => {
      this.categoriesFound = response.data
    })
    if (this.categoriesFound.length === 1 && this.categoryname === this.categoriesFound[0].CategoryName) {
      this.validCategory = true
    } else {
      this.validCategory = false
    }
  } catch (error) {
    console.log(error)
  }
  this.submitvalidated()
},
```

Anexo 39g

```
    async findPOIsByCityID (cityid) {
      const path = `http://localhost:5000/pois/filterby/cities?arg=` + cityid
      try {
        await axios.get(path, {
          headers: {
            Authorization: this.$parent.getToken()
          }
        })
          .then(response => {
            this.poisFound = response.data
          })
          .catch(error => {
            console.log(error)
          })
      } catch (error) {
        console.log(error)
      }
    }
  },
  created () {
    this.getCities()
    this.getCategories()
  }
}
</script>

<style scoped>
.error {
  color: red;
}
.success {
  color: green;
}
</style>
```

Anexo 40

```
import Api from '@/services/Api'

export default {
  findByIP (ip, cookie) {
    return Api().get('/cameras/findbyip?arg=' + ip.IPaddress, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 41

```
import Api from '@/services/Api'

export default {
  updateCamera (cameraInfo, cameraID, cookie) {
    return Api().put('/cameras/update?arg=' + cameraID, cameraInfo, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 42

```
import Api from '@/services/Api'

export default {
  deleteCamera (cameraID, cookie) {
    return Api().delete('/cameras/delete?arg=' + cameraID, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 43a

```
<template>
  <div>
    <br>
    <br>
    <h1 v-if="showlist">List Cameras</h1>
    <label v-if="showlist">Choose a camera from the list:
    <input list=results name="searchbar" v-model="searchbar" @change="findByIP" /></label>
    <b-button v-if="showlist" class="newButton" href="http://localhost:8080/camera/add">New...</b-button>
    <datalist v-if="showlist" id="results">
      <li v-for="camera in cameras" v-bind:key="camera.id">
        <option>{{camera.IPaddress}}</option>
      </li>
    </datalist>
    <br>
    <br>
    <table v-if="showlist" align="center">
      <tr>
        <td>ID</td>
        <td>IP Address</td>
        <td>POI</td>
      </tr>
      <tr v-for="camera in cameras" v-bind:key="camera.id">
        <td>
          {{camera.id}}
        </td>
        <td>
          {{camera.IPaddress}}
        </td>
        <td v-for="poi in pois" v-bind:key="poi.id" v-if="camera.POI_id === poi.id">
          {{poi.POIName}}
        </td>
        <td class="buttons">
          <b-button @click="updateButtonPressed(camera.id)">
            Update
          </b-button>
        </td>
        <td class="buttons">
          <b-button @click="deleteButtonPressed(camera.id)">
            Delete
          </b-button>
        </td>
      </tr>
    </table>
```

Anexo 43b

```
    <h1 v-if="showupdate">Edit Camera</h1>
    <input
      type="IPaddress"
      name="IPaddress"
      v-if="showupdate"
      v-model="cameraToUpdate.IPaddress"
      @input="validIP(cameraToUpdate.IPaddress)"
      :placeholder="cameraToUpdate.IPaddress"  />
      <br>
      <input list=poiresults v-if="showupdate" name="poiname" v-model="poiname" @focus="findByName" @input="findByName"/>
      <datalist id="poiresults">
        <li v-for="POIFound in pois" v-bind:key="POIFound.id">
          <option>{{POIFound.POIName}}</option>
        </li>
      </datalist>
      <br>
    <b-button v-if="showupdate" :disabled="!valid" @click="updateConfirmPressed(cameraToUpdate.id)">Confirm</b-button>
    <b-button v-if="showupdate" @click="back">Cancel</b-button>
    <div class="error" v-html="error" />
    <div class="success" v-html="success" />
    <div class="nomatches" v-html="nomatches" />
  </div>
</template>
```

Anexo 43c

```
<script>
import axios from 'axios'
import FindByIPService from '@/services/FindByIPService'
import FindByNameService from '@/services/FindByNameService'
import DeleteCameraService from '@/services/DeleteCameraService'
import UpdateCameraService from '@/services/UpdateCameraService'
var validateip = require('validate-ip')
export default {
  data () {
    return {
      cameras: [],
      cameraToUpdate: [],
      pois: [],
      searchbar: '',
      poiname: '',
      valid: true,
      success: null,
      error: null,
      result: null,
      nomatches: null,
      showlist: true,
      showupdate: false
    }
  },
  methods: {
    getCameras () {
      this.cameras = this.getCamerasFromBackend()
    },
    getCamerasFromBackend () {
      const path = `http://localhost:5000/cameras`
      axios.get(path, {
        headers: {
          Authorization: this.$parent.getToken()
        }
      })
        .then(response => {
          this.cameras = response.data
        })
        .catch(error => {
          console.log(error)
        })
    },
```

Anexo 43d

```
getPOIs () {
  this.pois = this.getPOIsFromBackend()
},
getPOIsFromBackend () {
  const path = `http://localhost:5000/pois`
  axios.get(path, {
    headers: {
      Authorization: this.$parent.getToken()
    }
  })
    .then(response => {
      this.pois = response.data
    })
    .catch(error => {
      console.log(error)
    })
},
async getCameraByIDFromBackend (cameraid) {
  const path = `http://localhost:5000/cameras/description?arg=` + cameraid
  try {
    await axios.get(path, {
      headers: {
        Authorization: this.$parent.getToken()
      }
    })
      .then(response => {
        this.cameraToUpdate = response.data[0]
      })
      .catch(error => {
        console.log(error)
      })
  } catch (error) {
    console.log('something happened')
  }
},
back () {
  window.location.reload()
},
```

Anexo 43e

```
deleteButtonPressed (cameraID) {
  var _this = this
  this.$dialog.confirm('Please confirm to continue')
    .then(function () {
      try {
        DeleteCameraService.deleteCamera(cameraID, _this.$parent.getToken())
      } catch (error) {
        console.log('not going to allow') // not working
      }
      window.location.reload()
    })
},
async updateButtonPressed (cameraID) {
  this.showlist = false
  console.log('button pressed')
  this.cameraToUpdate = this.getCameraByIDFromBackend(cameraID, this.$parent.getToken())
  this.showupdate = true
  console.log('success')
},
async updateConfirmPressed (cameraID) {
  try {
    if (this.pois.length === 1 && this.pois[0].POIName === this.poiname) {
      await UpdateCameraService.updateCamera({
        IPaddress: this.cameraToUpdate.IPaddress,
        POI_id: this.pois[0].id
      }, this.cameraToUpdate.id, this.$parent.getToken())
      window.location.reload()
      this.success = 'Updated Successfully'
    } else {
      this.error = 'please select a valid POI'
      this.success = null
    }
  } catch (error) {
    this.error = error.response.data.error
  }
},
```

Anexo 43f

```
async findByIP () {
  try {
    await FindByIPService.findByIP({
      IPaddress: this.searchbar
    }, this.$parent.getToken()).then(response => {
      this.cameras = response.data
    })
    if (this.cameras.length !== 0) {
      this.success = 'Found Successfully'
      this.error = null
      this.nomatches = null
    } else {
      this.error = null
      this.success = null
      this.nomatches = 'No matches found'
    }
  } catch (error) {
    this.error = 'Something happened'
    this.success = null
    this.nomatches = null
  }
},
async validIP (ip) {
  const validation = await validateip(ip)
  if (validation === true) {
    this.valid = true
  } else {
    this.valid = false
  }
},
```

Anexo 43g

```
    async findByName () {
      try {
        await FindByNameService.findByName({
          POIName: this.poiname
        }, this.$parent.getToken()).then(response => {
          this.pois = response.data
        })
        if (this.pois.length !== 0) {
          this.error = null
          this.nomatches = null
        } else {
          this.error = null
          this.success = null
          this.nomatches = 'No matches found'
        }
      } catch (error) {
        this.error = 'Something happened'
        this.success = null
        this.nomatches = null
      }
    }
  },
  created () {
    this.getCameras()
    this.getPOIs()
  }
}
</script>
```

Anexo 44

```
import Api from '@/services/Api'

export default {
  addCamera (cameraInfo, cookie) {
    return Api().post('/cameras/add', cameraInfo, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 45a

```
<template>
  <div>
  <br>
  <br>
    <h1>Add Camera</h1>

    <input
      type="IPaddress"
      name="IPaddress"
      v-model="IPaddress"
      placeholder="IPaddress"
      v-on:change="validIP(IPaddress)"
      v-on:input="validIP(IPaddress)" />
      <br>
      <input list=poiresults name="poiname" v-model="poiname" @focus="findByName" @input="findByName"/>
      <datalist id="poiresults">
        <li v-for="POIFound in pois" v-bind:key="POIFound.id">
          <option>{{POIFound.POIName}}</option>
        </li>
      </datalist>
      <br>
      <div class="error" v-html="error" />
      <div class="success" v-html="success" />
      <br>
      <b-button :disabled="!enableButton" @click="addCamera">
        Add Camera
      </b-button>
      <b-button @click="back">Back</b-button>
  </div>
</template>
```

Anexo 45b

```
<script>
import AddCameraService from '@/services/AddCameraService'
import FindByNameService from '@/services/FindByNameService'
var validateip = require('validate-ip')
export default {
  data () {
    return {
      IPaddress: '',
      POI_id: '',
      poiname: '',
      pois: [],
      error: null,
      validipaddress: false,
      validPOI: false,
      enableButton: false,
      success: null
    }
  },
  methods: {
    async addCamera () {
      try {
        await AddCameraService.addCamera({
          IPaddress: this.IPaddress,
          POI_id: this.pois[0].id
        }, this.$parent.getToken())
        this.success = 'Added Successfully'
        this.$router.push('../camera')
      } catch (error) {
        this.error = 'Please select an existing Point of Interest'
      }
    },
    async validIP (ip) {
      const validation = await validateip(ip)
      if (validation === true) {
        this.validipaddress = true
      } else {
        this.validipaddress = false
      }
      this.validateOutput()
    },
    back () {
      this.$router.push('../camera')
    },
```

Anexo 45c

```
    validateOutput () {
      if (this.validipaddress === true && this.validPOI === true) {
        this.enableButton = true
      } else {
        this.enableButton = false
      }
    },
    async findByName () {
      try {
        await FindByNameService.findByName({
          POIName: this.poiname
        }, this.$parent.getToken()).then(response => {
          this.pois = response.data
        })
        if (this.pois.length !== 0) {
          this.error = null
          this.nomatches = null
        } else {
          this.error = null
          this.success = null
          this.nomatches = 'No matches found'
        }
        if (this.pois.length === 1 && this.poiname === this.pois[0].POIName) {
          this.validPOI = true
        } else {
          this.validPOI = false
        }
        this.validateOutput()
      } catch (error) {
        this.error = 'Something happened'
        this.success = null
        this.nomatches = null
      }
    }
  }
}
</script>
```

Anexo 46

```
import Api from '@/services/Api'

export default {
  updateCategory (categoryInfo, categoryID, cookie) {
    return Api().put('/pois/categories/update?arg=' + categoryID, categoryInfo, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 47

```
import Api from '@/services/Api'

export default {
  deleteCategory (categoryID, cookie) {
    return Api().delete('/pois/categories/delete?arg=' + categoryID, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 48a

```
<template>
  <div>
    <br>
    <br>
    <h1 v-if="showlist">List Categories</h1>
    <label v-if="showlist">Choose a category from the list:
    <input list=results name="searchbar" v-model="searchbar" @change="findCategoryByName" /></label>
    <b-button v-if="showlist" class="newButton" href="http://localhost:8080/category/add">New...</b-button>
    <datalist v-if="showlist" id="results">
      <li v-for="categoryFound in categories" v-bind:key="categoryFound.id">
        <option>{{categoryFound.CategoryName}}</option>
      </li>
    </datalist>
    <br>
    <br>
    <table v-if="showlist" align="center">
      <tr>
        <td>ID</td>
        <td>Name</td>
      </tr>
      <tr v-for="category in categories" v-bind:key="category.id">
        <td>
          {{category.id}}
        </td>
        <td>
          {{category.CategoryName}}
        </td>
        <td class="buttons">
          <b-button @click="updateButtonPressed(category.id)">
            Update
          </b-button>
        </td>
        <td class="buttons">
          <b-button @click="deleteButtonPressed(category.id)">
            Delete
          </b-button>
        </td>
      </tr>
    </table>
```

Anexo 48b

```
    <h1 v-if="showupdate">Edit POI</h1>
    <input
      type="CategoryName"
      name="CategoryName"
      v-if="showupdate"
      v-model="categoryToUpdate.CategoryName"
      :placeholder="categoryToUpdate.CategoryName"  />
      <br>
    <input
      type="CategoryIconName"
      name="CategoryIconName"
      v-if="showupdate"
      v-model="categoryToUpdate.CategoryIconName"
      :placeholder="categoryToUpdate.CategoryIconName" />
    <br>
    <b-button v-if="showupdate" @click="updateConfirmPressed(categoryToUpdate.id)">Confirm</b-button>
    <b-button v-if="showupdate" @click="back">Cancel</b-button>
    <div class="error" v-html="error" />
    <div class="success" v-html="success" />
    <div class="nomatches" v-html="nomatches" />
  </div>
</template>
```

Anexo 48c

```
<script>
import axios from 'axios'
import FindCategoryByNameService from '@/services/FindCategoryByNameService'
import DeleteCategoryService from '@/services/DeleteCategoryService'
import UpdateCategoryService from '@/services/UpdateCategoryService'

export default {
  data () {
    return {
      categories: [],
      success: null,
      error: null,
      nomatches: null,
      showlist: true,
      showupdate: false,
      categoryToUpdate: [],
      searchbar: '',
      matches: [],
      results: []
    }
  },
  methods: {
    getCategories () {
      this.categories = this.getCategoriesFromBackend()
    },
    getCategoriesFromBackend () {
      const path = `http://localhost:5000/pois/categories`
      axios.get(path, {
        headers: {
          Authorization: this.$parent.getToken()
        }
      })
        .then(response => {
          this.categories = response.data
        })
        .catch(error => {
          console.log(error)
        })
    },
    back () {
      window.location.reload()
    },
```

Anexo 48d

```
deleteButtonPressed (categoryID) {
  var _this = this
  this.$dialog.confirm('Please confirm to continue')
    .then(function () {
      try {
        DeleteCategoryService.deleteCategory(categoryID, _this.$parent.getToken())
      } catch (error) {
        console.log(error)
      }
      window.location.reload()
    })
},
async updateButtonPressed (categoryID) {
  this.showlist = false
  this.categoryToUpdate = this.getCategoryByIDFromBackend(categoryID)
  this.showupdate = true
},
async findCategoryByName () {
  try {
    await FindCategoryByNameService.findCategoryByName({
      CategoryName: this.searchbar
    }, this.$parent.getToken()).then(response => {
      this.categories = response.data
      if (this.categories.length !== 0) {
        this.success = 'Found Successfully'
        this.error = null
        this.nomatches = null
      } else {
        this.error = null
        this.success = null
        this.nomatches = 'No matches found'
      }
    })
  } catch (error) {
    // add relog option
    console.log(error)
  }
},
```

Anexo 48e

```
    async getCategoryByIDFromBackend (categoryID) {
      const path = `http://localhost:5000/pois/categories/description?arg=` + categoryID
      try {
        await axios.get(path, {
          headers: {
            Authorization: `${this.$cookies.get('cdcToken')}`
          }
        })
          .then(response => {
            this.categoryToUpdate = response.data[0]
          })
          .catch(error => {
            console.log(error)
          })
      } catch (error) {
        console.log(error)
      }
    },
    async updateConfirmPressed (categoryID) {
      try {
        await UpdateCategoryService.updateCategory({
          CategoryName: this.categoryToUpdate.CategoryName,
          CategoryIconName: this.categoryToUpdate.CategoryIconName
        }, categoryID, this.$parent.getToken())
        window.location.reload()
      } catch (error) {
        console.log(error)
      }
      this.success = 'Updated Successfully'
    }
  },
  created () {
    this.getCategories()
  }
}
</script>
```

Anexo 49

```
import Api from '@/services/Api'

export default {
  addCategory (categoryInfo, cookie) {
    return Api().post('/pois/categories/add', categoryInfo, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 50a

```
<template>
  <div>
  <br>
  <br>
    <h1>Add Category</h1>
    <input
      type="CategoryName"
      name="CategoryName"
      v-model="CategoryName"
      placeholder="Category Name" />
    <br>
    <input
      type="CategoryIconName"
      name="CategoryIconName"
      v-model="CategoryIconName"
      placeholder="Category Icon Name" />
    <br>
    <br>
    <b-button @click="addCategory">
      Add Category
    </b-button>
    <b-button @click="back">Back</b-button>
  </div>
</template>
```

Anexo 50b

```
<script>
import AddCategoryService from '@/services/AddCategoryService'
export default {
  data () {
    return {
      CategoryName: '',
      CategoryIconName: ''
    }
  },
  methods: {
    async addCategory () {
      try {
        await AddCategoryService.addCategory({
          CategoryName: this.CategoryName,
          CategoryIconName: this.CategoryIconName
        }, this.$parent.getToken())
        this.$router.push('../category')
      } catch (error) {
        console.log(error)
      }
    },
    back () {
      this.$router.push('../category')
    }
  }
}
</script>
```

Anexo 51

```
import Api from '@/services/Api'

export default {
  updateCity (cityInfo, cityID, cookie) {
    return Api().put('/cities/update?arg=' + cityID, cityInfo, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 52

```
import Api from '@/services/Api'

export default {
  deleteCity (cityID, cookie) {
    return Api().delete('/cities/delete?arg=' + cityID, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 53a

```
<template>
  <div>
    <br>
    <br>
    <h1 v-if="showlist">List Cities</h1>
    <label v-if="showlist">Choose a city from the list:
    <input list=results name="searchbar" v-model="searchbar" @change="findCityByName" /></label>
    <b-button v-if="showlist" class="newButton" href="http://localhost:8080/city/add">New...</b-button>
    <datalist v-if="showlist" id="results">
      <li v-for="cityFound in cities" v-bind:key="cityFound.id">
        <option>{{cityFound.City}}</option>
      </li>
    </datalist>
    <br>
    <br>
    <table v-if="showlist" align="center">
      <tr>
        <td>ID</td>
        <td>Country</td>
        <td>City</td>
        <td>Geographical Latitude Center</td>
        <td>Geographical Longitude Center</td>
      </tr>
      <tr v-for="city in cities" v-bind:key="city.id">
        <td>
          {{city.id}}
        </td>
        <td>
          {{city.Country}}
        </td>
        <td>
          {{city.City}}
        </td>
        <td>
          {{city.geoLatCenter}}
        </td>
        <td>
          {{city.geoLongCenter}}
        </td>
        <td class="buttons">
          <b-button @click="updateButtonPressed(city.id)">
            Update
          </b-button>
        </td>
```

Anexo 53b

```
        <td class="buttons">
          <b-button @click="deleteButtonPressed(city.id)">
            Delete
          </b-button>
        </td>
      </tr>
    </table>
    <h1 v-if="showupdate">Edit City</h1>
    <input
      type="Country"
      name="Country"
      v-if="showupdate"
      v-model="cityToUpdate.Country"
      :placeholder="cityToUpdate.Country"/>
    <br>
    <input
      type="City"
      name="City"
      v-if="showupdate"
      v-model="cityToUpdate.City"
      :placeholder="cityToUpdate.City" />
    <br>
    <input
      type="geoLat"
      name="geoLat"
      v-if="showupdate"
      v-model="cityToUpdate.geoLatCenter"
      :placeholder="cityToUpdate.geoLatCenter"
      v-on:input="geoLatValidated()"/>
      <br>
    <input
      type="geoLong"
      name="geoLong"
      v-if="showupdate"
      v-model="cityToUpdate.geoLongCenter"
      :placeholder="cityToUpdate.geoLongCenter"
      v-on:input="geoLongValidated()"/>
    <br>
    <b-button v-if="showupdate" @click="updateConfirmPressed(cityToUpdate.id)">Confirm</b-button>
    <b-button v-if="showupdate" @click="back">Cancel</b-button>
    <div class="error" v-html="error" />
    <div class="success" v-html="success" />
    <div class="nomatches" v-html="nomatches" />
</div>
```

Anexo 53c

```
<script>
import axios from 'axios'
import UpdateCityService from '@/services/UpdateCityService'
import DeleteCityService from '@/services/DeleteCityService'
import FindCityByNameService from '@/services/FindCityByNameService'

export default {
  data () {
    return {
      cities: [],
      success: null,
      error: null,
      nomatches: null,
      showlist: true,
      showupdate: false,
      cityToUpdate: [],
      searchbar: '',
      results: [],
      validLat: false,
      validLong: false,
      validated: false
    }
  },
  methods: {
    getCities () {
      this.cities = this.getCitiesFromBackend()
    },
    getCitiesFromBackend () {
      const path = `http://localhost:5000/cities`
      axios.get(path, {
        headers: {
          Authorization: `${this.$cookies.get('cdcToken')}`
        }
      })
        .then(response => {
          this.cities = response.data
        })
        .catch(error => {
          console.log(error)
        })
    },
    back () {
      window.location.reload()
    },
```

Anexo 53d

```
geoLatValidated () {
  if (this.geoLatCenter > -90 && this.geoLatCenter < 90) {
    this.validLat = true
  } else {
    this.validLat = false
  }
  this.submitvalidated()
},
geoLongValidated () {
  if (this.geoLongCenter > -180 && this.geoLongCenter < 180) {
    this.validLong = true
  } else {
    this.validLong = false
  }
  this.submitvalidated()
},
submitvalidated () {
  if (this.validLat === true && this.validLong === true) {
    this.validated = true
  } else {
    this.validated = false
  }
},
deleteButtonPressed (cityID) {
  var _this = this
  this.$dialog.confirm('Please confirm to continue')
    .then(function () {
      try {
        DeleteCityService.deleteCity(cityID, _this.$parent.getToken())
      } catch (error) {
        console.log(error)
      }
      window.location.reload()
    })
},
async updateButtonPressed (cityID) {
  this.showlist = false
  this.cityToUpdate = this.getCityByIDFromBackend(cityID)
  this.showupdate = true
},
```

Anexo 53e

```
async getCityByIDFromBackend (cityID) {
  const path = `http://localhost:5000/cities/description?arg=` + cityID
  try {
    await axios.get(path, {
      headers: {
        Authorization: this.$parent.getToken()
      }
    })
      .then(response => {
        this.cityToUpdate = response.data[0]
      })
      .catch(error => {
        console.log(error)
      })
  } catch (error) {
    console.log(error)
  }
},
async findCityByName () {
  try {
    await FindCityByNameService.findCityByName({
      City: this.searchbar
    }, this.$parent.getToken()).then(response => {
      this.cities = response.data
      if (this.cities.length !== 0) {
        this.success = 'Found Successfully'
        this.error = null
        this.nomatches = null
      } else {
        this.error = null
        this.success = null
        this.nomatches = 'No matches found'
      }
    })
  } catch (error) {
    // add relog option
    console.log(error)
  }
},
```

Anexo 53f

```
    async updateConfirmPressed (cityID) {
      try {
        await UpdateCityService.updateCity({
          City: this.cityToUpdate.City,
          Country: this.cityToUpdate.Country,
          geoLatCenter: this.cityToUpdate.geoLatCenter,
          geoLongCenter: this.cityToUpdate.geoLongCenter
        }, cityID, this.$parent.getToken())
        window.location.reload()
      } catch (error) {
        console.log(error)
      }
      this.success = 'Updated Successfully'
    }
  },
  created () {
    this.getCities()
  }
}
</script>
```

Anexo 54

```
import Api from '@/services/Api'

export default {
  addCity (cityInfo, cookie) {
    return Api().post('/cities/add', cityInfo, {
      headers: {
        Authorization: cookie
      }
    })
  }
}
```

Anexo 55a

```
<template>
  <div>
  <br>
  <br>
    <h1>Add City</h1>
    <input
      type="CountryName"
      name="CountryName"
      v-model="Country"
      placeholder="Country Name" />
    <br>
    <input
      type="CityName"
      name="CityName"
      v-model="City"
      placeholder="City Name" />
    <br>
    <input
      type="geoLat"
      name="geoLat"
      v-model="geoLatCenter"
      placeholder="Geo-center Latitude"
      v-on:input="geoLatValidated()"/>
      <br>
    <input
      type="geoLong"
      name="geoLong"
      v-model="geoLongCenter"
      placeholder="Geo-center Longitude"
      v-on:input="geoLongValidated()"/>
    <br>
    <b-button :disabled="!validated" @click="addCity">
      Add City
    </b-button>
    <b-button @click="back">Back</b-button>
  </div>
</template>
```

Anexo 55b

```
<script>
import AddCityService from '@/services/AddCityService'
export default {
  data () {
    return {
      City: '',
      Country: '',
      geoLatCenter: '',
      geoLongCenter: '',
      validLat: false,
      validLong: false,
      validated: false
    }
  },
  methods: {
    async addCity () {
      try {
        await AddCityService.addCity({
          City: this.City,
          Country: this.Country,
          geoLatCenter: this.geoLatCenter,
          geoLongCenter: this.geoLongCenter
        }, this.$parent.getToken())
        this.$router.push('../city')
      } catch (error) {
        console.log(error)
      }
    },
    back () {
      this.$router.push('../city')
    },
    geoLatValidated () {
      if (this.geoLatCenter > -90 && this.geoLatCenter < 90) {
        this.validLat = true
      } else {
        this.validLat = false
      }
      this.submitvalidated()
    },
```

Anexo 55c

```
    geoLongValidated () {
      if (this.geoLongCenter > -180 && this.geoLongCenter < 180) {
        this.validLong = true
      } else {
        this.validLong = false
      }
      this.submitvalidated()
    },
    submitvalidated () {
      if (this.validLat === true && this.validLong === true) {
        this.validated = true
      } else {
        this.validated = false
      }
    }
  }
}
</script>
```

Anexo 56a

```
<template>
  <div>
  <br>
  <br>
    <h1>List Predictions</h1>
    <table align="center">
      <tr>
        <td>ID</td>
        <td>Prediction</td>
        <td>Camera ID</td>
        <td>Created at</td>
      </tr>
      <tr v-for="prediction in predictions" v-bind:key="prediction.id">
        <td>
          {{prediction.id}}
        </td>
        <td>
          {{prediction.Prediction}}
        </td>
        <td>
          {{prediction.Camera_id}}
        </td>
        <td>
          {{prediction.Time}}
        </td>
      </tr>
    </table>
  <br>
  <div class="error" v-html="error" />
  <div class="success" v-html="success" />
  <div class="nomatches" v-html="nomatches" />
  </div>
</template>
```

Anexo 56b

```
<script>
import axios from 'axios'

export default {
  data () {
    return {
      predictions: [],
      success: null,
      error: null,
      result: null,
      nomatches: null
    }
  },
  methods: {
    getPredictions () {
      this.pois = this.getPredictionsFromBackend()
    },
    getPredictionsFromBackend () {
      const path = `http://localhost:5000/predictions`
      axios.get(path, {
        headers: {
          Authorization: this.$parent.getToken()
        }
      })
        .then(response => {
          this.predictions = response.data
        })
        .catch(error => {
          console.log(error)
        })
    }
  },
  created () {
    this.getPredictions()
  }
}
</script>
```

Anexo 57a

```
<template>
  <div>
    <br>
    <br>
    <h1 v-if="showlist">List Users</h1>
    <label v-if="showlist">Choose a user from the list:
    <input list=results name="searchbar" v-model="searchbar" @change="findUserByName" /></label>
    <datalist v-if="showlist" id="results">
      <li v-for="userFound in users" v-bind:key="userFound.id">
        <option>{{userFound.email}}</option>
      </li>
    </datalist>
    <br>
    <br>
    <table v-if="showlist" align="center">
      <tr>
        <td>ID</td>
        <td>Email</td>
        <td>Role</td>
      </tr>
      <tr v-for="user in users" v-if="user.id!==1" v-bind:key="user.id">
        <td>
          {{user.id}}
        </td>
        <td>
          {{user.email}}
        </td>
        <td>
          {{user.role}}
        </td>
        <td class="buttons">
          <b-button @click="updateButtonPressed(user.id)">
            Update Role
          </b-button>
        </td>
        <td class="buttons">
          <b-button @click="deleteButtonPressed(user.id)">
            Delete
          </b-button>
        </td>
      </tr>
    </table>
```

Anexo 57b

```
    <h1 v-if="showupdate">Edit user</h1>
    <select v-if="showupdate" v-model="newRole">
      <option disabled value="">Please select one</option>
      <option value="user">User</option>
      <option value="admin">Admin</option>
    </select>
    <br>
    <b-button v-if="showupdate" @click="updateConfirmPressed(userToUpdate.id)">Confirm</b-button>
    <b-button v-if="showupdate" @click="back">Cancel</b-button>
  </div>
</template>
```

Anexo 57c

```
<script>
import axios from 'axios'
import DeleteUserService from '@/services/DeleteUserService'
import FindUserByNameService from '@/services/FindUserByNameService'
import UpdateUserService from '@/services/UpdateUserService'

export default {
  data () {
    return {
      users: [],
      showlist: true,
      searchbar: '',
      showupdate: false,
      success: null,
      error: null,
      nomatches: null,
      results: [],
      newRole: ''
    }
  },
  methods: {
    getUsers () {
      this.users = this.getUsersFromBackend()
    },
    getUsersFromBackend () {
      const path = `http://localhost:5000/users`
      axios.get(path, {
        headers: {
          Authorization: `${this.$cookies.get('cdcToken')}`
        }
      })
        .then(response => {
          this.users = response.data
        })
        .catch(error => {
          console.log(error)
        })
    },
    back () {
      window.location.reload()
    },
```

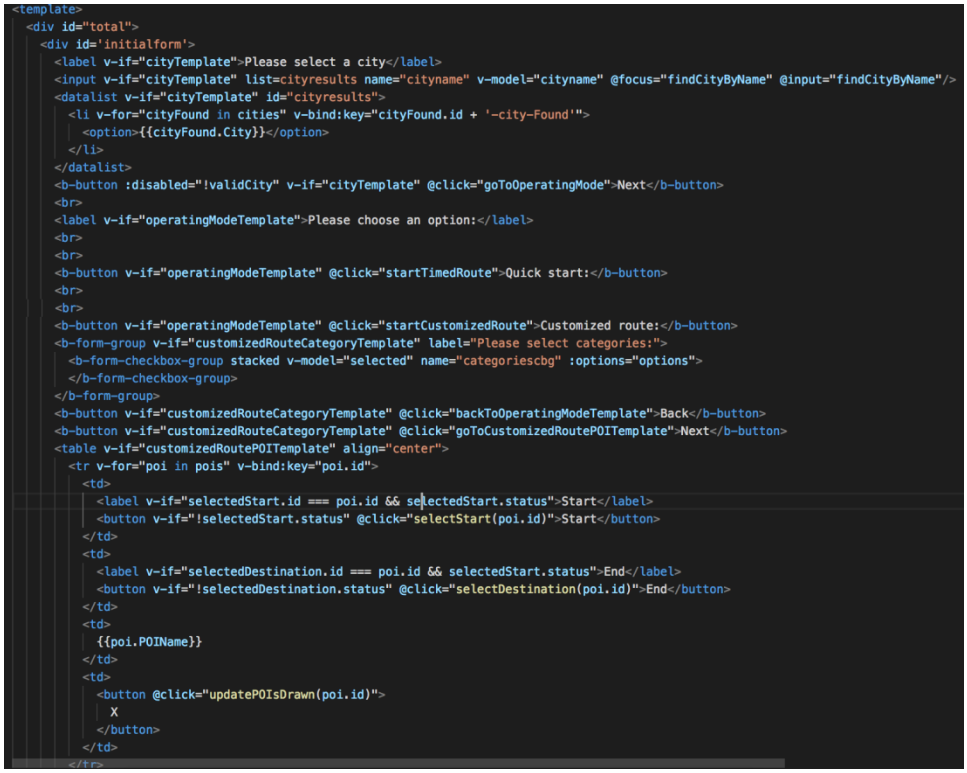
Anexo 57d

```
deleteButtonPressed (userID) {
  var _this = this
  this.$dialog.confirm('Please confirm to continue')
    .then(function () {
      try {
        DeleteUserService.deleteUser(userID, _this.$parent.getToken())
      } catch (error) {
        console.log(error)
      }
      window.location.reload()
    })
},
async updateButtonPressed (userID) {
  this.showlist = false
  this.userToUpdate = this.getUserByIDFromBackend(userID)
  this.showupdate = true
},
async getUserByIDFromBackend (userID) {
  const path = `http://localhost:5000/users/description?arg=` + userID
  try {
    await axios.get(path, {
      headers: {
        Authorization: this.$parent.getToken()
      }
    })
      .then(response => {
        this.userToUpdate = response.data[0]
      })
      .catch(error => {
        console.log(error)
      })
  } catch (error) {
    console.log(error)
  }
},
```

Anexo 57e

```
    async updateConfirmPressed (userID) {
      try {
        console.log(this.newRole)
        await UpdateUserService.updateUser({
          id: this.userToUpdate.id,
          email: this.userToUpdate.email,
          password: this.userToUpdate.password,
          role: this.newRole
        }, userID, this.$parent.getToken())
        window.location.reload()
      } catch (error) {
        console.log(error)
      }
      this.success = 'Updated Successfully'
    },
    async findUserByName () {
      try {
        await FindUserByNameService.findUserByName({
          email: this.searchbar
        }, this.$parent.getToken()).then(response => {
          this.users = response.data
          if (this.categories.length !== 0) {
            this.success = 'Found Successfully'
            this.error = null
            this.nomatches = null
          } else {
            this.error = null
            this.success = null
            this.nomatches = 'No matches found'
          }
        })
      } catch (error) {
        // add relog option
        console.log(error)
      }
    }
  },
  created () {
    this.getUsers()
  }
}
</script>
```

Anexo 58a



```
<template>
  <div id="total">
    <div id='initialform'>
      <label v-if="cityTemplate">Please select a city</label>
      <input v-if="cityTemplate" list=cityresults name="cityname" v-model="cityname" @focus="findCityByName" @input="findCityByName"/>
      <datalist v-if="cityTemplate" id="cityresults">
        <li v-for="cityFound in cities" v-bind:key="cityFound.id + '-city-Found'">
          <option>{{cityFound.City}}</option>
        </li>
      </datalist>
      <b-button :disabled="!validCity" v-if="cityTemplate" @click="goToOperatingMode">Next</b-button>
      <br>
      <label v-if="operatingModeTemplate">Please choose an option:</label>
      <br>
      <br>
      <b-button v-if="operatingModeTemplate" @click="startTimedRoute">Quick start:</b-button>
      <br>
      <br>
      <b-button v-if="operatingModeTemplate" @click="startCustomizedRoute">Customized route:</b-button>
      <b-form-group v-if="customizedRouteCategoryTemplate" label="Please select categories:">
        <b-form-checkbox-group stacked v-model="selected" name="categoriescbg" :options="options">
        </b-form-checkbox-group>
      </b-form-group>
      <b-button v-if="customizedRouteCategoryTemplate" @click="backToOperatingModeTemplate">Back</b-button>
      <b-button v-if="customizedRouteCategoryTemplate" @click="goToCustomizedRoutePOITemplate">Next</b-button>
      <table v-if="customizedRoutePOITemplate" align="center">
        <tr v-for="poi in pois" v-bind:key="poi.id">
          <td>
            <label v-if="selectedStart.id === poi.id && selectedStart.status">Start</label>
            <button v-if="!selectedStart.status" @click="selectStart(poi.id)">Start</button>
          </td>
          <td>
            <label v-if="selectedDestination.id === poi.id && selectedStart.status">End</label>
            <button v-if="!selectedDestination.status" @click="selectDestination(poi.id)">End</button>
          </td>
          <td>
            {{poi.POIName}}
          </td>
          <td>
            <button @click="updatePOIsDrawn(poi.id)">
              X
            </button>
          </td>
        </tr>
```

Anexo 58b

```
      </table>
      <b-button v-if="customizedRoutePOITemplate" @click="backToCustomizedCategoryTemplate">Back</b-button>
      <b-button v-if="customizedRoutePOITemplate" :disabled="!validFinish" @click="startRoute">Finish</b-button>
    </div>
    <div id='map'></div>
  </div>
</template>

<script>
import mapboxgl from 'mapbox-gl'
import axios from 'axios'
import FindCityByNameService from '@/services/FindCityByNameService'
import GetDensityService from '@/services/GetDensityService'
require('../../node_modules/mapbox-gl/dist/mapbox-gl.css')
var $ = require('jQuery')
// constructing geoJSON
var geojsonPOIs = {
  id: 'pois',
  type: 'symbol',
  source: {
    type: 'geojson',
    data: {
      type: 'FeatureCollection',
      features: []
    }
  },
  layout: {
    'icon-image': '{icon}-15',
    'icon-allow-overlap': true
  }
}
```

Anexo 58c

```
export default {
  data () {
    return {
      // variables inialized within the context
      map: null,
      apiKey: 'pk.eyJ1IjoiaWNsaW8iLCJhIjoiY2l2bnByem9pMDAwajJucDh5MWFiOTUwdCJ9.RNHNU1MujwhDoHknx31_OQ',
      cityTemplate: true,
      operatingModeTemplate: false,
      customizedRouteCategoryTemplate: false,
      customizedRoutePOITemplate: false,
      geoJSONPOIs: geojsonPOIs,
      userDefinedDuration: 200,
      start: [],
      distancesToStart: {
        id: null,
        distance: null
      },
      destination: [],
      selectedStart: {
        status: false,
        id: null
      },
      selectedDestination: {
        status: false,
        id: null
      },
      currentPosition: {
        latitude: null,
        longitude: null
      },
```

Anexo 58d

```
      validFinish: false,
      cities: [],
      categories: [],
      pois: [],
      citiesFound: [],
      categoriesFound: [],
      selected: [],
      options: [],
      selectedPOIs: [],
      POIoptions: [],
      cityname: '',
      city_id: null,
      categoryname: '',
      validCity: false,
      validCategory: false,
      error: null,
      success: null,
      finalMatrix: [],
      waitList: [],
      accumulatedTime: 0,
      bestCombo: [],
      layersAdded: 0,
      finalDistance: 0,
      finalDuration: 0,
      minHistoricDistance: 99999999999,
      maxHistoricPointsVisited: 0,
      bestHighPriorityCombo: []
    }
  },
  methods: {
    // get the list of cities
    getCities () {
      this.cities = this.getCitiesFromBackend()
    },
```

Anexo 58e

```
getCitiesFromBackend () {
  const path = `http://localhost:5000/cities`
  axios.get(path, {
    headers: {
      Authorization: `${this.$cookies.get('cdcToken')}`
    }
  }).then(response => {
      this.cities = response.data
    })
    .catch(error => {
      console.log(error)
    })
},
// find cities from the user's input
async findCityByName () {
  var _this = this
  try {
    await FindCityByNameService.findCityByName({
      City: this.cityname
    }, _this.$parent.getToken()).then(response => {
      this.citiesFound = response.data
    })
    if (this.citiesFound.length === 1 && this.cityname === this.citiesFound[0].City) {
      this.validCity = true
    } else {
      this.validCity = false
    }
  } catch (error) {
    console.log(error)
  }
  // this.submitvalidated()
},
// inicialize map object
createMap () {
  mapboxgl.accessToken = this.apiKey
  // init the map
  this.map = new mapboxgl.Map({
    container: 'map',
    style: 'mapbox://styles/iclio/cjf8bwvxt4wlg2rqp3a9e9xid',
    center: [-8.222, 40.223],
    zoom: 13
  })
},
```

Anexo 58f

```
// inicialize map object
createMap () {
  mapboxgl.accessToken = this.apiKey
  // init the map
  this.map = new mapboxgl.Map({
    container: 'map',
    style: 'mapbox://styles/iclio/cjf8bwvxt4wlg2rqp3a9e9xid',
    center: [-8.222, 40.223],
    zoom: 13
  })
},
// get Categories from backend
getCategories () {
  this.getCategoriesFromBackend()
},
getCategoriesFromBackend () {
  const path = `http://localhost:5000/pois/categories`
  axios.get(path, {
    headers: {
      Authorization: `${this.$cookies.get('cdcToken')}`
    }
  })
    .then(response => {
      this.categories = response.data
    })
    .catch(error => {
      console.log(error)
    })
},
// change from city template to operating mode template
goToOperatingMode () {
  this.map.flyTo({center: [this.citiesFound[0].geoLongCenter, this.citiesFound[0].geoLatCenter]})
  this.cityTemplate = false
  this.getDistanceMatrix(this.citiesFound[0].id)
  this.operatingModeTemplate = true
},
```

Anexo 58g

```
async startTimedRoute () {
  var poiData = []
  var _this = this
  const path = `http://localhost:5000/pois/findbycity?arg=` + this.citiesFound[0].id
  try {
    await axios.get(path, {
      headers: {
        Authorization: this.$parent.getToken()
      }
    })
      .then(response => {
        this.pois = response.data
      })
      .catch(error => {
        console.log(error)
      })
  } catch (error) {
    console.log(error)
  }
  this.openList = this.pois
  for (var listIterator in this.openList) {
    poiData.push({id: this.openList[listIterator].id})
  }
  var _this = this
  try {
    await GetDensityService.getDensity({
      poiList: poiData
    }, _this.$parent.getToken()).then(response => {
      this.densitiesFound = response.data
    })
  } catch (error) {
    console.log(error)
  }
  var highPriorityList = []
  var lowPriorityList = []
```

Anexo 58h

```
// remove points with max density from OpenList and add them to the Wait List
for (var pointDensity in this.densitiesFound) {
  if (this.densitiesFound[pointDensity] === 'high_density') {
    this.waitList.push(Number(pointDensity))
  } else {
    var test = this.getPriority(Number(pointDensity))
    if (test < 4) {
      lowPriorityList.push(Number(pointDensity))
    }else {
      highPriorityList.push(Number(pointDensity))
    }
  }
}
this.openList = highPriorityList
console.log('wait list', this.waitList)
console.log('low priority list', lowPriorityList)
var minDistToStart = 999999999
var firstPointID
var firstPointDistance
for (var highPriorityListIterator in highPriorityList){
  await this.getDistanceToStart(highPriorityList[highPriorityListIterator])
  if(this.distancesToStart.distance < minDistToStart) {
    minDistToStart = this.distancesToStart.distance
    firstPointID = this.distancesToStart.id
    firstPointDistance = this.distancesToStart.distance
  }
}
var accumulatedDistance = this.distancesToStart.distance
var accumulatedTime = (accumulatedDistance * 0.8 / 60)
console.log('first point', this.distancesToStart.id)
this.bestCombo.push(this.distancesToStart.id)
this.openList.splice(this.openList.indexOf(this.distancesToStart.id),1)
console.log(this.openList)
console.log(accumulatedTime)
var stopTime = this.userDefinedDuration * 2 / 3
var pointToLock = 0
var done = false
var previousPoint = this.distancesToStart.id
var bestPoint
var walkingDistance
```

Anexo 58i

```
while(!done){
  pointToLock = pointToLock + 1
  var indexesToRemove = []
  for (var waitPoint in this.waitList) {
    var AvgVisitTime = this.getAvgVisitTime(this.waitList[waitPoint])
    if (accumulatedTime > AvgVisitTime){
      var priority = this.getPriority(this.waitList[waitPoint])
      if( priority >= 4) {
        this.openList.push(this.waitList[waitPoint])
        console.log('added point ', this.waitList[waitPoint], ' to high priority list')
      }else {
        lowPriorityList.push(this.waitList[waitPoint])
        console.log('added point ', this.waitList[waitPoint], ' to low priority list')
      }
      indexesToRemove.push(this.waitList[waitPoint])
      console.log('removed point ', this.waitList[waitPoint])
    }
  }
  var entryToRemove = indexesToRemove.length - 1
  while(entryToRemove >= 0){
    this.waitList.splice(this.waitList.indexOf(indexesToRemove[entryToRemove]))
    entryToRemove = entryToRemove - 1
  }
  var minDistance = 9999999
  for (var point in this.openList){
    console.log('point', this.openList[point])
    walkingDistance = this.getDistanceFromMatrix(previousPoint, this.openList[point])
    console.log('distance to point',walkingDistance)
    if(walkingDistance < minDistance){
      minDistance = walkingDistance
      bestPoint = this.openList[point]
    }
  }
  this.bestCombo.push(bestPoint)
  previousPoint = bestPoint
  accumulatedDistance = accumulatedDistance + minDistance
  var updatedTime = accumulatedTime + (minDistance * 0.8 / 60) + this.getAvgVisitTime(bestPoint)
  this.openList.splice(this.openList.indexOf(bestPoint),1)
```

Anexo 58j

```
  if(this.openList.length === 0 || updatedTime > stopTime){
    done = true
    console.log('hit time limit')
  }else {
    accumulatedTime = updatedTime
  }
}
var tempCombo = []

console.log('best combo', this.bestCombo)
console.log('accumulated distance', accumulatedDistance)
console.log('accumulated time', accumulatedTime)
console.log('low priority points to explore', lowPriorityList)
var listOfPointsToAdd = []
var pointToAdd = {
  id: 0,
  position: '',
  timeDifference: 0
}
for(var lowPriorityPoint in lowPriorityList){
  var closestPoint = this.getClosestPoint(lowPriorityList[lowPriorityPoint])
  var VisitTimeForLowPriorityPoint = this.getAvgVisitTime(lowPriorityList[lowPriorityPoint])
  var counter = 0
  while(counter < this.bestCombo.length){
    if(closestPoint = this.bestCombo[counter]){
      if(counter > 0){
        var currentDistanceAddBefore = this.getDistanceFromMatrix(this.bestCombo[counter], this.bestCombo[counter-1])
        var newDistanceAddBefore = this.getDistanceFromMatrix(this.bestCombo[counter-1],lowPriorityList[lowPriorityPoint]) + this.getDistanceFromMatr
        var timeDifferenceBefore = ((newDistanceAddBefore - currentDistanceAddBefore) * 0.8) + VisitTimeForLowPriorityPoint
      }
      if(counter < this.bestCombo.length - 1){
        var currentDistanceAddAfter = this.getDistanceFromMatrix(this.bestCombo[counter], this.bestCombo[counter+1])
        var newDistanceAddAfter = this.getDistanceFromMatrix(this.bestCombo[counter],lowPriorityList[lowPriorityPoint]) + this.getDistanceFromMatrix(
        var timeDifferenceAfter = ((newDistanceAddAfter - currentDistanceAddAfter) * 0.8) + VisitTimeForLowPriorityPoint
      }
    }
    counter = counter + 1
  }
```

Anexo 58k

```
  if(timeDifferenceBefore === null && timeDifferenceAfter === null) {
    console.log('not adding anywhere')
  }
  else if(timeDifferenceBefore !== null){
    if(timeDifferenceAfter!== null){
      if(timeDifferenceBefore<timeDifferenceAfter){
        pointToAdd = {
          id: lowPriorityList[lowPriorityPoint],
          position: 'before',
          timeDifference: timeDifferenceBefore,
          nextTo: closestPoint
        }
        console.log('point to add', pointToAdd)
      }else {
        pointToAdd = {
          id: lowPriorityList[lowPriorityPoint],
          position: 'after',
          timeDifference: timeDifferenceAfter,
          nextTo: closestPoint
        }
        console.log('point to add', pointToAdd)
      }
    }else {
      pointToAdd = {
        id: lowPriorityList[lowPriorityPoint],
        position: 'before',
        timeDifference: timeDifferenceBefore,
        nextTo: closestPoint
      }
      console.log('point to add', pointToAdd)
    }
  }else if(timeDifferenceAfter !== null){
    pointToAdd = {
      id: lowPriorityList[lowPriorityPoint],
      position: 'after',
      timeDifference: timeDifferenceAfter,
      nextTo: closestPoint
    }
    console.log('point to add', pointToAdd)
  }
  listOfPointsToAdd.push(pointToAdd)
}
var addCounter = 0
```

Anexo 58l

```
var visitedPoints = []
var orderedList = []
while(addCounter < listOfPointsToAdd.length) {
  var minTimeDifference = 99999999
  var minTimeID
  var minTimePosition
  var minTimeNextTo
  for (var singlePoint in listOfPointsToAdd){
    if(visitedPoints.includes(listOfPointsToAdd[singlePoint].id) === false){
      if(listOfPointsToAdd[singlePoint].timeDifference < minTimeDifference){
        console.log('new min')
        minTimeDifference = listOfPointsToAdd[singlePoint].timeDifference
        minTimeID = listOfPointsToAdd[singlePoint].id
        minTimePosition = listOfPointsToAdd[singlePoint].position
        minTimeNextTo = listOfPointsToAdd[singlePoint].nextTo
      }
    }
    console.log('visited Point', minTimeID)
  }
  console.log('push')
  orderedList.push({id:minTimeID, position: minTimePosition, timeDifference: minTimeDifference, nextTo: minTimeNextTo})
  visitedPoints.push(minTimeID)
  addCounter = addCounter + 1
}
console.log(accumulatedTime)
console.log(orderedList)
console.log('time remaining', this.userDefinedDuration - accumulatedTime)
```

Anexo 58m

```
var finalCombo = []
for(var pointAdded in orderedList){
  console.log('came here')
  var updatedTimeOfAddedPoint = accumulatedTime + orderedList[pointAdded].timeDifference
  if(updatedTimeOfAddedPoint < this.userDefinedDuration ){
    for(var pointFinder in this.bestCombo){
      if(this.bestCombo[pointFinder]===orderedList[pointAdded].nextTo){
        if(orderedList[pointAdded].position === 'before'){
          if(finalCombo.includes(orderedList[pointAdded].id) === false) {
            finalCombo.push(orderedList[pointAdded].id)
            finalCombo.push(this.bestCombo[pointFinder])
          }
        } else  if(orderedList[pointAdded].position === 'after'){
          if(finalCombo.includes(orderedList[pointAdded].id) === false) {
            finalCombo.push(this.bestCombo[pointFinder])
            finalCombo.push(orderedList[pointAdded].id)
          }
        }
      }else {
        if(finalCombo.includes(this.bestCombo[pointFinder]) === false) {
          finalCombo.push(this.bestCombo[pointFinder])
        }
      }
    }
  }else {
    console.log('not adding any')
    finalCombo = this.bestCombo
    break
  }
}
console.log('final result', finalCombo)
var promises = []
var indexesToRemove = []
```

Anexo 58n

```
for( var tempiterator in this.pois){
  var vardelete = true
  for (var i in finalCombo) {
    if(this.pois[tempiterator].id === finalCombo[i]){
      vardelete = false
    }
  }
  if(vardelete === true){
    indexesToRemove.push(this.pois.indexOf(this.pois[tempiterator]))
  }
}
console.log(indexesToRemove)
var tempIterator = indexesToRemove.length - 1
while (tempIterator > -1) {
  this.pois.splice(indexesToRemove[tempIterator], 1)
  tempIterator = tempIterator - 1
}
Promise.all(promises)
  .then(result => {
    this.drawPOIs()
  })
console.log('done')
this.finalDistance = accumulatedDistance
this.finalDuration = accumulatedTime
var previousPoint
var promises = []
var firstIter = true
for (point in finalCombo) {
  if (firstIter) {
    firstIter = false
    promises.push(this.drawDistanceToUserLocation(this.currentPosition.latitude,this.currentPosition.longitude,finalCombo[point]))
    previousPoint = finalCombo[point]
  } else {
    promises.push(this.drawDirections(previousPoint, finalCombo[point], this.layersAdded))
    console.log('drawing from ', previousPoint, ' to ', finalCombo[point])
    previousPoint = finalCombo[point]
    this.layersAdded = this.layersAdded + 1
  }
}
console.log(finalCombo)
console.log(accumulatedDistance)
console.log(accumulatedTime)
```

Anexo 58o

```
    // draw optimal route
    Promise.all(promises)
      .then(function () {
        console.log('done')
      })
  },
  getClosestPoint(poiID){
    var tempDist
    var minDist = 9999999
    var bestPoint
    for(var point in this.bestCombo){
      tempDist = this.getDistanceFromMatrix(this.bestCombo[point],poiID)
      if(tempDist < minDist ){
        minDist = tempDist
        bestPoint = this.bestCombo[point]
      }
    }
    return bestPoint
  },
```

Anexo 58p

```
async drawDistanceToUserLocation (userLocationLat, userLocationLong, endPoint) {
  var endlatitude
  var endlongitude
  for (var poiIterator in this.pois) {
    if (this.pois[poiIterator].id === endPoint) {
      endlongitude = this.pois[poiIterator].geoLong
      endlatitude = this.pois[poiIterator].geoLat
    }
  }
  var directionsRequest = 'https://api.mapbox.com/directions/v5/mapbox/walking/' + userLocationLong + ',' + userLocationLat + ';' + endlongitude + ','
  var _this = this
  try {
    await $.ajax({
      method: 'GET',
      url: directionsRequest
    }).done(function (data) {
      var route = data.routes[0]
      _this.finalDuration = _this.finalDuration + route.duration
      _this.finalDistance = _this.finalDistance + route.distance
      _this.map.addLayer({
        id: 'startRoute',
        type: 'line',
        source: {
          type: 'geojson',
          lineMetrics: true,
          data: {
            type: 'Feature',
            geometry: route.geometry
          }
        },
        paint: {
          'line-width': 2,
          'line-color': "cyan"
        },
        layout: {
          'line-cap': 'round',
          'line-join': 'round'
        }
      })
```

Anexo 58q

```
        _this.map.addLayer({
          id: 'startRoutearrows',
          type: 'symbol',
          source: 'startRoute',
          layout: {
            'symbol-placement': 'line',
            'text-field': '▶',
            'text-size': {
              base: 1,
              stops: [[12, 24], [22, 60]]
            },
            'symbol-spacing': {
              base: 1,
              stops: [[12, 30], [22, 160]]
            },
            'text-keep-upright': false
          },
          paint: {
            'text-color': '#3887be'
          }
        })
      })
    }catch (error) {
      console.log(error)
    }
  },
  // go to Categories template
  startCustomizedRoute () {
    this.operatingModeTemplate = false
    this.customizedRouteCategoryTemplate = true
    this.updateCategoryOptions()
  },
  // go back to operating mode template
  backToOperatingModeTemplate () {
    this.customizedRouteCategoryTemplate = false
    this.operatingModeTemplate = true
  },
```

Anexo 58r

```
// gets all POIs and draws them on map
goToCustomizedRoutePOITemplate () {
  var promises = []
  for (var i in this.selected) {
    promises.push(this.filterPOIsByCategory(this.selected[i]))
  }
  Promise.all(promises)
    .then(result => {
      this.drawPOIs()
      this.customizedRouteCategoryTemplate = false
      this.customizedRoutePOITemplate = true
      this.updatePOIOptions()
    })
},
// go back to customized category
backToCustomizedCategoryTemplate () {
  this.customizedRoutePOITemplate = false
  this.customizedRouteCategoryTemplate = true
},
// category checkbox options
updateCategoryOptions () {
  this.options = []
  for (var singleCat in this.categories) {
    this.options.push({
      text: this.categories[singleCat].CategoryName,
      value: this.categories[singleCat].id
    })
  }
},
// poi checkbox options
updatePOIOptions () {
  for (var singlepoi in this.pois) {
    this.POIoptions.push({
      text: this.pois[singlepoi].POIName,
      value: this.pois[singlepoi].id
    })
  }
  for (var singlePOIoption in this.POIoptions) {
    this.selectedPOIs.push(this.POIoptions[singlePOIoption].value)
  }
},
```

Anexo 58s

```
filterPOIsByCategory (catID) {
  const path = `http://localhost:5000/pois/filterby/categories?arg=` + catID
  return axios.get(path, {
    headers: {
      Authorization: `${this.$cookies.get('cdcToken')}`
    }
  })
    .then(response => {
      for (var count in response.data) {
        this.pois.push(response.data[count])
      }
    })
    .catch(error => {
      console.log(error)
    })
},
// select the start point of the route
selectStart (poiID) {
  for (var singPOI in this.pois) {
    if (this.pois[singPOI].id === poiID) {
      var temp = this.geoJSONPOIs.source.data.features[singPOI]
      this.geoJSONPOIs.source.data.features[singPOI] = this.geoJSONPOIs.source.data.features[0]
      this.geoJSONPOIs.source.data.features[0] = temp
      this.selectedStart.status = true
      this.selectedStart.id = poiID
      break
    }
  }
  this.checkValidFinish()
},
// select the destination point of the route
selectDestination (poiID) {
  for (var singPOI in this.pois) {
    if (this.pois[singPOI].id === poiID) {
      var temp = this.geoJSONPOIs.source.data.features[singPOI]
      this.geoJSONPOIs.source.data.features[singPOI] = this.geoJSONPOIs.source.data.features[this.geoJSONPOIs.source.data.features.length - 1]
      this.geoJSONPOIs.source.data.features[this.geoJSONPOIs.source.data.features.length - 1] = temp
      this.selectedDestination.status = true
      this.selectedDestination.id = poiID
      break
    }
  }
  this.checkValidFinish()
},
```

Anexo 58t

```
// validates if start and destination points are set
checkValidFinish () {
  if (this.selectedStart.status === true && this.selectedDestination.status === true) {
    this.validFinish = true
  } else {
    this.validFinish = false
  }
},
// remove unselected pois from list and route
deselect (poiID) {
  for (var singPOI in this.pois) {
    if (this.pois[singPOI].id === poiID) {
      this.pois.splice(singPOI, 1)
      break
    }
  }
  this.removeMapInfo()
},
// remove previous map info if user goes back in templates to avoid information overwrite
removeMapInfo () {
  this.map.removeLayer('pois')
  this.map.removeSource('pois')
  this.geoJSONPOIs.source.data.features = []
},
```

Anexo 58u

```
// draws pois selected by the user
drawPOIs () {
  for (var i = 0; i < this.pois.length; i++) {
    var iconName = this.getCategoryIconFromID(this.pois[i].Category_id)
    this.geoJSONPOIs.source.data.features.push({
      type: 'Feature',
      geometry: {
        type: 'Point',
        coordinates: [this.pois[i].geoLong, this.pois[i].geoLat]
      },
      properties: {
        icon: iconName,
        description: this.pois[i].POIName
      }
    })
  }
  this.map.addLayer(this.geoJSONPOIs)
  var _this = this
  // set on click options and popups
  this.map.on('click', 'pois', function (e) {
    var coordinates = e.features[0].geometry.coordinates.slice()
    var description = e.features[0].properties.description
    while (Math.abs(e.lngLat.lng - coordinates[0]) > 180) {
      coordinates[0] += e.lngLat.lng > coordinates[0] ? 360 : -360
    }
    new mapboxgl.Popup()
      .setLngLat(coordinates)
      .setHTML(description)
      .addTo(_this.map)
  })
},
// get a category icon from  category id
getCategoryIconFromID (catID) {
  for (var singleCat in this.categories) {
    if (this.categories[singleCat].id === catID) {
      return this.categories[singleCat].CategoryIconName
    }
  }
},
```

Anexo 58v

```
// gets distance matrix from database
getDistanceMatrix (cityID) {
  const path = `http://localhost:5000/cities/distanceMatrix?arg=` + cityID
  return axios.get(path, {
    headers: {
      Authorization: `${this.$cookies.get('cdcToken')}`
    }
  })
    .then(response => {
      this.distanceMatrix = response.data
      this.finalMatrix = $.parseJSON(this.distanceMatrix)
    })
    .catch(error => {
      console.log(error)
    })
},
// returns the distance in between 2 points
getDistanceFromMatrix (point1, point2) {
  var matrixKey
  if (point1 < point2) {
    matrixKey = '' + String(point1) + '-' + String(point2) // will be useful to get distances later
  } else {
    matrixKey = '' + String(point2) + '-' + String(point1) // will be useful to get distances later
  }
  return this.finalMatrix[matrixKey]
},
```

Anexo 58w

```
async getDistanceToStart (poiID) {
  // get the distance from this.currentPosition to the lat and log of the poi with id = poiID
  for (var singlePOI in this.pois) {
    if(this.pois[singlePOI].id === poiID) {
      var _this = this
      var directionsRequest = 'https://api.mapbox.com/directions/v5/mapbox/walking/' + _this.currentPosition.longitude  + ',' + _this.currentPosition.l
      try {
        await $.ajax({
          method: 'GET',
          url: directionsRequest
        }).done(function (data) {
          var route = data.routes[0]
          _this.distancesToStart.id = poiID
          _this.distancesToStart.distance = route.distance
        })
      } catch (error) {
        console.log(error)
      }
    }
  }
},
// routing algorithm
async startRoute () {
  // add all points to Open List
  this.openList = []
  for (var singlepoi in this.selectedPOIs) {
    this.openList.push(this.selectedPOIs[singlepoi])
  }
  // remove start and destination from Open List
  var start = this.selectedStart.id
  var destination = this.selectedDestination.id
  for (var point in this.openList) {
    if (this.openList[point] === start) {
      this.openList.splice(point, 1)
    }
    if (this.openList[point] === destination) {
      this.openList.splice(point, 1)
    }
  }
```

Anexo 58x

```
// get all poi ids
var poiData = []
for (var listIterator in this.openList) {
  poiData.push({id: this.openList[listIterator]})
}
var _this = this
// use the api service to get the density of the points of interest
try {
  await GetDensityService.getDensity({
    poiList: poiData
  }, _this.$parent.getToken()).then(response => {
    this.densitiesFound = response.data
  })
} catch (error) {
  console.log(error)
}
// remove points with max density from OpenList and add them to the Wait List
for (var pointDensity in this.densitiesFound) {
  if (this.densitiesFound[pointDensity] === 'high_density') {
    var tempIndex = this.openList.indexOf(Number(pointDensity))
    this.waitList.push(this.openList[tempIndex])
    this.openList.splice(tempIndex, 1)
  }
}
// get all permutations of the openList
var combos = this.permute(this.openList)
// inicialize support variables for the route algorithm
this.bestCombo.push(this.selectedStart.id)
var accumulatedDistance = 0
var pointToLock = 0
var previousPoint = this.selectedStart.id
var tempWaitTimeforPoi
var NNAlgorithm = true
var AStarAlgorithm = false
var modified = false
```

Anexo 58y

```
while (pointToLock < this.openList.length && combos.length > 0) {
  var waitTimeWithKeys = {}
  for (var waitPointIndex in this.waitList) {
    waitTimeWithKeys[this.waitList[waitPointIndex]] = this.getAvgVisitTime(this.waitList[waitPointIndex])
  }
  for (var key in waitTimeWithKeys) {
    if (waitTimeWithKeys[key] < this.accumulatedTime) {
      this.openList.push(Number(key))
      this.waitList.splice(this.waitList.indexOf(key), 1)
      modified = true
    }
  }
  if (modified) {
    combos = this.UpdateOpenList()
    modified = false
    if (this.waitList.length === 0) {
      NNAlgorithm = false
      AStarAlgorithm = true
    }
  }
  // ------------------------------------
  if (NNAlgorithm) {
    var ignore
    if (combos.length === 1) {
      ignore = true
    }
    var pointsChecked = []
    var minDistance = 999999999
    var bestPoint
    for (var comboIterator in combos) {
      if (!pointsChecked.includes(combos[comboIterator][pointToLock]) && !this.bestCombo.includes(combos[comboIterator][pointToLock])) {
        var tempDist = this.getDistanceFromMatrix(combos[comboIterator][pointToLock], previousPoint)
        if (minDistance > tempDist) {
          minDistance = tempDist
          bestPoint = combos[comboIterator][pointToLock]
        }
        pointsChecked.push(combos[comboIterator][pointToLock])
      }
    }
    accumulatedDistance = accumulatedDistance + minDistance
    tempWaitTimeforPoi = this.getAvgVisitTime(bestPoint)
    this.accumulatedTime = this.accumulatedTime + ((0.8 * minDistance) / 60) + tempWaitTimeforPoi // assuming people walk at 0.8 m/s, cal
```

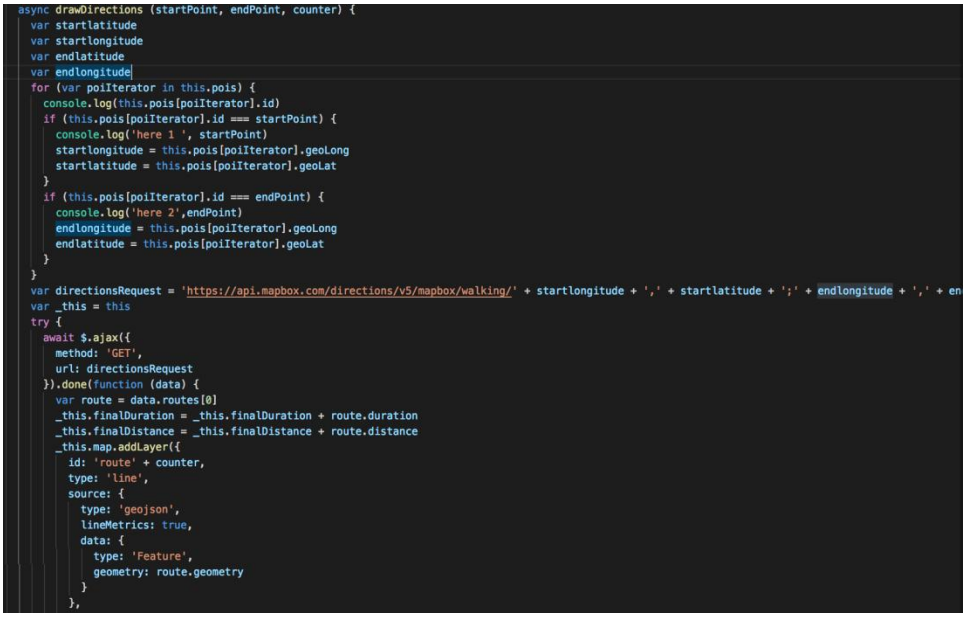
Anexo 58za

```
  if (!ignore) {
    this.bestCombo.push(bestPoint)
    var indexesToRemove = []
    for (var comboI in combos) {
      if (combos[comboI][pointToLock] !== bestPoint) {
        indexesToRemove.push(combos.indexOf(combos[comboI]))
      }
    }
    var tempIterator = indexesToRemove.length - 1
    while (tempIterator > -1) {
      combos.splice(indexesToRemove[tempIterator], 1)
      tempIterator = tempIterator - 1
    }
  }
  if (ignore) {
    // autocomplete the remaining best combination with the only combination left on the list
    var counter = 1
    for (var autocompleteIterator in combos[0]) {
      if (this.bestCombo[counter] !== combos[0][autocompleteIterator]) {
        this.bestCombo.push(combos[0][autocompleteIterator])
      }
      counter = counter + 1
    }
    pointToLock = 9999 // finish algorithm outer while loop
  }
}
// ------------------------------------
if (AStarAlgorithm) {
  var tempBestCombo
  var minDistance = 9999999999
  for (var comboIteratorAStar in combos) {
    var distance = 0
    previousPoint = combos[comboIteratorAStar][pointToLock - 1]
    for (var point in combos[comboIteratorAStar]) {
      if(previousPoint !== combos[comboIteratorAStar][point]){
        distance = distance + this.getDistanceFromMatrix(previousPoint, combos[comboIteratorAStar][point])
        previousPoint = combos[comboIteratorAStar][point]
        }
    }
    if(distance < minDistance) {
      minDistance = distance
      tempBestCombo = combos[comboIteratorAStar]
    }
```

Anexo 58zb

```
    var test = this.bestCombo[this.bestCombo.length - 1]
    var AStarAutocompleteIterator = tempBestCombo.indexOf(test) + 1
    while(AStarAutocompleteIterator < tempBestCombo.length) {
      this.bestCombo.push(tempBestCombo[AStarAutocompleteIterator])
      AStarAutocompleteIterator = AStarAutocompleteIterator + 1
    }
    pointToLock = 9999 // exit outer while loop
  }

  // ------------------------------------
  pointToLock = pointToLock + 1
  previousPoint = bestPoint
}
this.bestCombo.push(this.selectedDestination.id)
previousPoint = -1
var promises = []
var firstIter = true
for (point in this.bestCombo) {
  if (firstIter) {
    firstIter = false
    previousPoint = this.bestCombo[point]
  } else {
    promises.push(this.drawDirections(previousPoint, this.bestCombo[point], this.layersAdded))
    previousPoint = this.bestCombo[point]
    this.layersAdded = this.layersAdded + 1
  }
}
// update final duration to add the AvgVisitTime of each POI
for (var durationUpdateIterator in this.bestCombo){
  this.finalDuration = this.finalDuration + (this.getAvgVisitTime(this.bestCombo[durationUpdateIterator]) * 60)
}
// draw optimal route
_this = this
Promise.all(promises)
  .then(function () {
    console.log('best route is : ', _this.bestCombo)
    var hours = Math.floor(_this.finalDuration / 3600)
    var tempRemainer = _this.finalDuration % 3600
    var minutes = Math.floor(tempRemainer / 60)
    var seconds = Math.floor(tempRemainer % 60)
    console.log('with a duration of ', hours + ' h : ' + minutes + ' m : ' + seconds + 's')
    console.log('and a distance of : ', (_this.finalDistance/1000).toFixed(1) + ' km ')
  })
```

Anexo 58zc



```
async drawDirections (startPoint, endPoint, counter) {
  var startlatitude
  var startlongitude
  var endlatitude
  var endlongitude
  for (var poiIterator in this.pois) {
    console.log(this.pois[poiIterator].id)
    if (this.pois[poiIterator].id === startPoint) {
      console.log('here 1 ', startPoint)
      startlongitude = this.pois[poiIterator].geoLong
      startlatitude = this.pois[poiIterator].geoLat
    }
    if (this.pois[poiIterator].id === endPoint) {
      console.log('here 2',endPoint)
      endlongitude = this.pois[poiIterator].geoLong
      endlatitude = this.pois[poiIterator].geoLat
    }
  }
  var directionsRequest = 'https://api.mapbox.com/directions/v5/mapbox/walking/' + startlongitude + ',' + startlatitude + ';' + endlongitude + ',' + en
  var _this = this
  try {
    await $.ajax({
      method: 'GET',
      url: directionsRequest
    }).done(function (data) {
      var route = data.routes[0]
      _this.finalDuration = _this.finalDuration + route.duration
      _this.finalDistance = _this.finalDistance + route.distance
      _this.map.addLayer({
        id: 'route' + counter,
        type: 'line',
        source: {
          type: 'geojson',
          lineMetrics: true,
          data: {
            type: 'Feature',
            geometry: route.geometry
          }
        },
```

Anexo 58zd

```
          layout: {
            'line-cap': 'round',
            'line-join': 'round'
          }
        })
        _this.map.addLayer({
          id: 'routearrows' + counter,
          type: 'symbol',
          source: 'route' + counter,
          layout: {
            'symbol-placement': 'line',
            'text-field': '▶',
            'text-size': {
              base: 1,
              stops: [[12, 24], [22, 60]]
            },
            'symbol-spacing': {
              base: 1,
              stops: [[12, 30], [22, 160]]
            },
            'text-keep-upright': false
          },
          paint: {
            'text-color': '#3887be'
          }
        })
      })
    } catch (error) {
      console.log('start', startPoint, 'end', endPoint)
    }

  },
```

Anexo 58ze

```
permute (permutation) {
  // gets all permutations
  var length = permutation.length
  var result = [permutation.slice()]
  var c = new Array(length).fill(0)
  var i = 1
  var k
  var p
  while (i < length) {
    if (c[i] < i) {
      k = i % 2 && c[i]
      p = permutation[i]
      permutation[i] = permutation[k]
      permutation[k] = p
      ++c[i]
      i = 1
      result.push(permutation.slice())
    } else {
      c[i] = 0
      ++i
    }
  }
  return result
},
UpdateOpenList () {
  var indexesToRemove = []
  var tempCounter = 0
  var combos = this.permute(this.openList)
  while (tempCounter < this.bestCombo.length) {
    for (var comboIterator in combos) {
      if (Number(combos[comboIterator][tempCounter]) !== Number(this.bestCombo[tempCounter + 1]) && !indexesToRemove.includes(comboIterator)) {
        indexesToRemove.push(comboIterator)
      }
    }
    var tempIterator = indexesToRemove.length - 1
    while (tempIterator > -1) {
      combos.splice(indexesToRemove[tempIterator], 1)
      tempIterator = tempIterator - 1
    }
    tempCounter = tempCounter + 1
  }
  return combos
},
```

Anexo 58zf

```
    getAvgVisitTime (poiID) {
      for (var poi in this.pois) {
        if (this.pois[poi].id === poiID) {
          return this.pois[poi].AvgVisitTime
        }
      }
    },
    getPriority (poiID) {
      for (var poi in this.pois) {
        if(this.pois[poi].id === poiID) {
          return this.pois[poi].priority
        }
      }
    },
    async getLocation () {
      try {
        var position = await this.getCurrentPosition();
        var { latitude, longitude } = position.coords;
        this.currentPosition = {
          latitude,
          longitude
        };
      } catch (error) {
        console.log(error);
      }
    },
    getCurrentPosition(options = {}) {
      return new Promise((resolve, reject) => {
        navigator.geolocation.getCurrentPosition(resolve, reject, options);
      })
    }
  },
  created () {
    this.getCities()
    this.getCategories()
    this.getLocation()
  },
  mounted () {
    this.createMap()
  }
}
</script>
```