

Escola de Engenharia
Universidade do Minho

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

Laboratórios de Informática IV

Plug&GoBeyond
(Relatório)



Cecília Soares, A34900



Gonçalo Esteves, A85731



João Linhares, A86618



Joel Ferreira, A89982



Luís Abreu, A82888

Docente:
Hugo Peixoto

19 de Fevereiro de 2020

Conteúdo

1	Introdução	1
2	Apresentação do Projecto	2
2.1	Estado da Arte	2
2.2	Funcionalidades	3
2.3	Objectivos e Problemas a tratar	4
2.4	Valor Acrescentado da Aplicação Desenvolvida	5
3	Modelação da Solução	6
3.1	Use Cases e Modelo de Domínio	6
3.2	Construção do Diagrama de Classes	8
3.3	Base de Dados	12
3.3.1	Modelo Lógico da Base de Dados	13
3.3.2	Modelo Físico	15
4	Implementação da Solução Adoptada	16
4.1	Arquitectura	16
4.1.1	Restful API	16
4.1.2	Front-End	19
4.1.3	Back-End	39
5	Conclusão	47
6	Anexos	49
6.1	Anexo 1	49
6.1.1	Use case: Registar utilizador	49
6.1.2	Use case: Registar viatura	50
6.1.3	Use case: Iniciar Sessão	51
6.1.4	Use case: Terminar sessão	51
6.1.5	Use case: Adicionar posto de carregamento favorito	51
6.1.6	Use case: Planear Viagem	52
6.1.7	Use case: Seleccionar veículo como favorito numa determinada sessão	53
6.1.8	Use case: Pesquisar Postos	53

6.1.9	Use case: Alterar <i>Password</i>	54
6.1.10	Use case: Adicionar Cartão de Crédito	55
6.1.11	Use case: Apagar Conta	55
6.1.12	Use case: Obter direcções para a estação mais próxima . .	56
6.1.13	Use case: Obter direcções para uma determinada estação	56
6.1.14	Use case: Consultar Perfil	57
6.1.15	Use case: Consultar Viagens Realizadas	57
6.1.16	Use case: Consultar Cartões de Crédito	58
6.1.17	Use case: Consultar Veículos	58
6.1.18	Use case: Consultar Postos Favoritos	58
6.1.19	Use case: Remover Viagem	59
6.1.20	Use case: Remover Cartão de Crédito	59
6.1.21	Use case: Remover Veículo	60
6.1.22	Use case: Remover Posto dos Favoritos	60
6.2	Anexo 2	62
6.3	Anexo 3	66
6.3.1	Diagrama de Gantt	66

Capítulo 1

Introdução

O presente projecto foi desenvolvido no âmbito da disciplina de Laboratórios de Informática 4, leccionada no 3.º ano do curso de MIEI, e destinou-se à criação, concepção e desenvolvimento de um produto mobile, com valor acrescentado face ao contexto de mercado actual, entre outras orientações genéricas que teríamos de obedecer.

Face ao desafio proposto, a equipa de trabalho decidiu criar uma aplicação mobile designada **Plug&GoBeyond**, dedicada ao planeamento de viagens e carregamento de automóveis eléctricos.

A escolha recaiu sobre esta temática por dois principais factores. Em primeiro lugar, devido ao crescente investimento do sector automóvel nos carros eléctricos e na franca expansão deste mercado, o que faz com que desenvolver produtos relacionados com este sector se mostre oportuno, aliciante e com interesse a nível económico. O segundo factor prende-se com a existência de algumas lacunas nas aplicações disponibilizadas aos condutores. Na verdade, o principal problema que identificamos foi a inexistência de uma aplicação mobile que reunisse todas as funcionalidades e características indispensáveis para um maior conforto do condutor. Efectivamente, depois de uma análise do mercado, verificamos que nenhuma aplicação mobile concatena as funcionalidades de planear uma viagem (o que implica indicar ao condutor o preço, o percurso e os postos de carregamento a serem visitados, bem como a duração da viagem e as tomadas a serem utilizadas para o carregamento) e de obter informações acerca de postos de carregamento, tal como localização, direcções, etc.

O presente relatório está organizado do seguinte modo: no capítulo 2 faz-se a apresentação do projecto, onde analisamos o estado da arte, as funcionalidades da aplicação, os objectivos e problemas a tratar, bem como o valor acrescentado da nossa *app*. No capítulo 3 descrevemos a modelação arquitectural do nosso projecto, onde definimos os use cases, o modelo de domínio e a nossa Base de Dados. No capítulo 4 descrevemos a implementação da solução adoptada, onde nos debruçamos sobre a arquitectura do nosso programa, a *RestAPI* utilizada e as propriedades do *front-end* e do *back-end*. Por último, no capítulo 5 apresentamos uma análise crítica do resultado e propostas de trabalho futuro.

Capítulo 2

Apresentação do Projecto

As questões ambientais e a preocupação com a defesa do ambiente é um tema recorrente na sociedade actual. Aliado a esta questão temos ainda o elevado e oscilante custo do combustível fóssil e a escassez a médio, longo prazo do mesmo. Todas estas razões levam a que haja um investimento significativo e crescente na indústria automóvel em busca do mais viável substituto para o petróleo.

Assim, o grupo de trabalho entendeu que o desenvolvimento de uma aplicação *mobile*, designada **Plug&GoBeyond**, dedicada ao planeamento de viagens e carregamento de automóveis eléctricos, com vista a facilitar a vida dos condutores, seria oportuna e adequada ao contexto actual.

Nessa medida, com o presente trabalho pretende-se desenvolver um *software* que disponibilize as informações necessárias quanto aos carregamentos, preços, potências de tomadas, localização de postos de carregamento, como ainda a possibilidade do utilizador saber as necessidades de carregamento, custo e trajecto a efectuar para se deslocar a determinado destino e, finalmente, a opção do condutor poder identificar as estações de carregamento favoritas.

Por último, convém mencionar que para o desenvolvimento do presente projecto, o grupo de trabalho adoptou um processo iterativo e incremental e de forma a gerir o projecto e a planear as várias etapas e o tempo gasto em cada uma delas criamos o diagrama de Gantt que consta do Anexo 3.

2.1 Estado da Arte

O crescente investimento da indústria automóvel no fabrico e diversificação dos modelos dos veículos eléctricos repercutiu-se nas vendas dos mesmos. Em 2018, foram vendidos, em Portugal, 8241 viaturas eléctricas, sendo Portugal, à data, o sexto país do mundo que mais vende estes veículos¹. Esta tendência manteve-se e, somente no primeiro trimestre de 2020, foram vendidos 4777 automóveis

¹ Informação retirada do site <https://observador.pt/2019/05/09/portugal-atinge-recorde-de-vendas-de-veiculos-eletricos-em-2018-e-e-o-sexto-pais-mais-vendedor-do-mundo/>

eléctricos², o que faz com que Portugal ocupe, actualmente, o quarto lugar nas vendas de viaturas elétricas.³ Ora, esta tendência abriu a porta para a criação de várias aplicações que pretendem facilitar o dia-a-dia dos condutores dos carros eléctricos e uma maior interacção entre essa comunidade. Em Portugal, existem algumas aplicações que permitem ao utilizador obter informações acerca do estado e ocupação dos postos de carregamento, do preço dos carregamentos e planejar viagens. Exemplos disso são aplicações como a *miio*, *MOB.I.E*, *A Better Route Planner (ABRP)* e a *ChargeMap*, as quais permitem ao condutor conhecer o estado e ocupação dos postos de carregamento, o preço dos carregamentos, os tipos dos mesmos, obter notificações, em que postos que será necessário abastecer durante um itinerário, registar novos postos de carregamento, possibilidade de os utilizadores inserirem fotos e comentários acerca de um posto, etc.

Com efeito, a panóplia de funcionalidades e serviços disponíveis nas mais variadas aplicações móveis têm como objectivo proporcionar a realização de uma viagem com o conforto e segurança necessários sem precisar de dispensar nenhum tempo no planeamento da mesma.

Nessa medida, a nossa aplicação apresenta-se como uma alternativa eficaz para a concretização deste objectivo, visto que agrega as funcionalidades de identificação dos postos de carregamento, bem como a possibilidade de planejar uma viagem, ao contrário das demais. Com efeito, nenhuma das aplicações reúne as funcionalidades de identificação dos postos de carregamento e de planejar viagens.

2.2 Funcionalidades

Nesta primeira fase, o grupo de trabalho resolveu focar-se no desenvolvimento das seguintes funcionalidades que ficarão disponíveis para o sistema operativo mobile *Android*:

- a exibição de um mapa geográfico que indique a localização de um posto de carregamento no território português;
- registar um utilizador;
- registar veículos automóveis associados a um utilizador;
- a possibilidade de planejar uma viagem com destino dentro da Europa;
- identificar determinados postos como favoritos;
- identificar determinado veículo automóvel do utilizador como favorito;
- pesquisar postos de carregamento;

²Informação retirada do site <https://observador.pt/2020/05/26/vendas-de-automoveis-eletricos-aumentam-mas-covid-19-pode-comprometer-setor-revela-estudo/>

³<https://www.uve.pt/page/blueauto-junho-2020-vendas-ve-resistem-hecatombe/>

- alterar password;
- adicionar cartões de crédito à conta do utilizador;
- apagar a conta de um utilizador;
- obter direcções para o posto de carregamento mais próximo através do *GoogleMaps*;
- obter direcções para um posto de carregamento específico através do *GoogleMaps*;
- Consultar perfil do utilizador;
- Consultar as viagens realizadas e informações associadas;
- Consultar os veículos de um utilizador;
- Consultar os cartões de crédito de um utilizador;
- Consultar os postos favoritos de um utilizador;
- Eliminar uma viagem, um cartão de crédito, um veículo automóvel ou um posto de carregamento da lista de favoritos;

2.3 Objectivos e Problemas a tratar

Ao desenvolver a aplicação referida deparamo-nos com alguns problemas no que se refere à recolha de informação relativa à localização dos postos de carregamento, tipo de carregamento possível, quantidade de tomadas, potência das mesmas, consumo dos automóveis consoante as marcas, modelos, etc.

No que se refere à localização dos postos de carregamento, conseguimos uma API do mapa de postos de carregamento e informação relativa às suas tomadas no site da *OpenChargeMap*⁴, a qual integramos com a nossa aplicação.

O segundo problema foi conseguir obter toda a informação necessária para planear uma viagem, nomeadamente o consumo por Km de cada automóvel eléctrico. Essa informação foi obtida graças aos dados disponibilizados pela empresa *Iternio*, que teve a amabilidade de nos ajudar ao fornecer-nos a sua API. Com a integração dessa API, o desenvolvimento da funcionalidade de planear uma viagem tornou-se bastante mais simples, pois a referida API tem acesso às características dos carros eléctricos, bem como aos postos de carregamento, sua localização e características das suas tomadas.

Assim, ao utilizarmos esta API conseguimos saber quais as estações de carregamento em que o condutor deve parar para chegar ao destino, a duração de cada carregamento, a melhor rota até ao destino e o custo desta deslocação. Tudo isto sem termos de gerir todos estes diferentes factores que influenciam o tratamento a dar à rotina de planear uma viagem.

⁴<https://openchargemap.org/site/develop/api>

2.4 Valor Acrescentado da Aplicação Desenvolvida

A mais-valia do presente projecto centra-se, primordialmente, em dois factores.

Em primeiro lugar, a *App* que agora apresentamos combina uma série de funcionalidades existentes no mercado, mas que nenhuma outra aplicação reúne. De facto, não ignoramos que a maioria das funcionalidades aqui apresentadas e por nós desenvolvidas já existem e são já amplamente utilizadas. Todavia, a inovação que introduzimos foi a reunião de todas elas.

Com efeito, a novidade está em a nossa aplicação disponibilizar informação acerca de todos os postos de carregamento e ainda permitir ao condutor planejar uma viagem, conseguindo saber a duração, o custo, os postos de carregamento que vai utilizar até chegar ao destino e o percurso mais conveniente até ao destino tendo em conta as paragens que terá de efectuar. Com a combinação destas duas funcionalidades, a nossa aplicação mobile revela-se bastante conveniente, oferecendo um leque de opções que garantem maior conforto e flexibilidade aos condutores de veículos eléctricos.

Acresce que a nossa aplicação permite que um condutor obtenha indicação do percurso até determinado posto de carregamento através da aplicação *GoogleMaps*.

Finalmente, convém referir que a nossa aplicação é compatível com os diversos sistemas operativos e a arquitectura por nós definida permite que a mesma seja facilmente actualizada e de evolução fácil e simples.

Capítulo 3

Modelação da Solução

A modelação arquitectural do nosso projecto começou com a utilização das ferramentas de desenvolvimento de software como a descrição de *Use Cases*, a construção do nosso *Modelo de Domínio* e os *Diagramas de Classes*.

3.1 Use Cases e Modelo de Domínio

Em primeiro lugar, definimos as situações que o nosso programa deveria tratar e caracterizamos, de uma forma muito genérica, como deveria ser efectuado esse tratamento. Desta feita, chegamos à descrição dos *Use Cases* contida em anexo, Anexo 1, bem como ao diagrama geral de *Use Cases* especificado na figura 3.1.

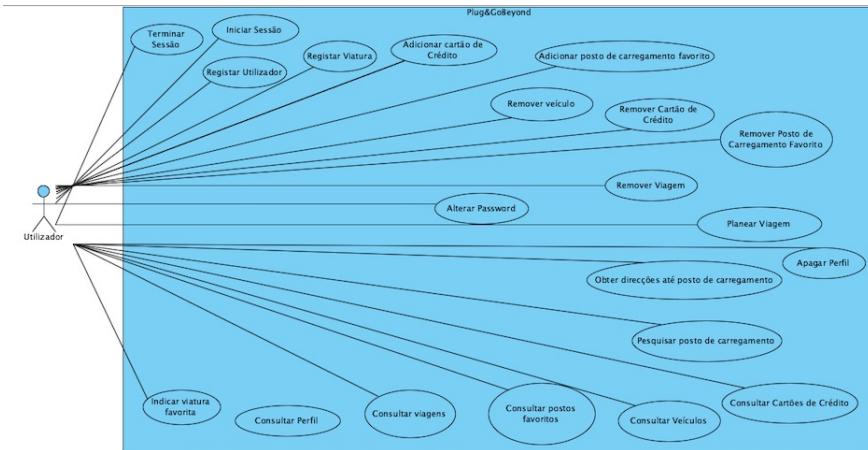


Figura 3.1: Diagrama de Use Cases

De acordo com o diagrama apresentado, podemos observar que nos debruçamos sobre 21 situações elementares, as quais estão descritas no diagrama

de *Use Cases* apresentado, bem como nas descrições de *Use Cases* prevista no Anexo 1.

Em primeiro lugar, seria fundamental inserir no nosso programa os intervenientes principais, ou seja, o utilizador, os postos de carregamento e os automóveis, pois sem estes o efeito prático da *app* é nulo.

Em segundo lugar, pensamos nas ações principais que o sistema deveria ter, nomeadamente iniciar e encerrar sessão, marcar um posto como favorito e planear uma viagem.

Depois de definirmos as funções básicas do nosso sistema e os actores que interaciam com o sistema, criamos o nosso modelo de domínio, o qual se pode observar na figura 3.2.

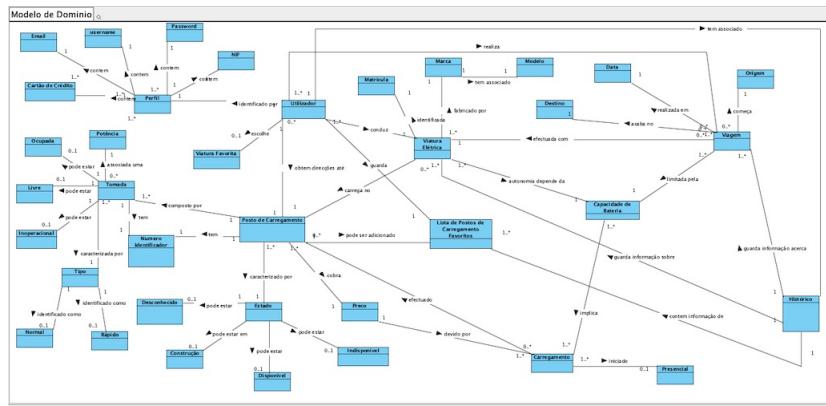


Figura 3.2: Modelo de Domínio

Conforme podemos observar no nosso modelo de domínio, o utilizador tem um perfil composto por cinco elementos que caracterizam e identificam um usuário da aplicação. Assim como o utilizador um posto de carregamento tem um estado que o caracteriza quanto à sua disponibilidade de funcionamento, podendo assim estar disponível, indisponível, em construção ou ter o seu estado desconhecido. Também as tomadas podem ser do tipo rápido ou normal, sendo que isso indica o tipo de carregamento que pode ser feito.

Além das características associadas ao utilizador, aos postos e às tomadas, indicamos os elementos que caracterizam o utilizador, os veículos e as viagens.

Ademais, o modelo de domínio indica a relação entre os diferentes actores do sistema e dá uma perspectiva geral do sistema que queremos implementar.

3.2 Construção do Diagrama de Classes

Após análise cuidada e atenta dos requisitos identificados para a nossa aplicação, e com o intuito de tornar o seu desenvolvimento mais simples e eficaz, recorremos ao uso da linguagem a UML - Unified Modeling Language para desenhar o classes diagrama de classes da aplicação.

Foram por nós detectadas como classes principais:

- ChargingStation
- StationWithoutValidation
- Trips
- User
- Vehicle Models
- Vehicle

Estas são as classes basilares da nossa aplicação, e permitir-nos-ão responder aos pedidos do utilizador. Os pedidos possíveis de realizar pelo utilizador, podem ser visualizados nas figuras seguintes, sendo que o diagrama da classe User encontra-se dividido em duas imagens para uma melhor compreensão.

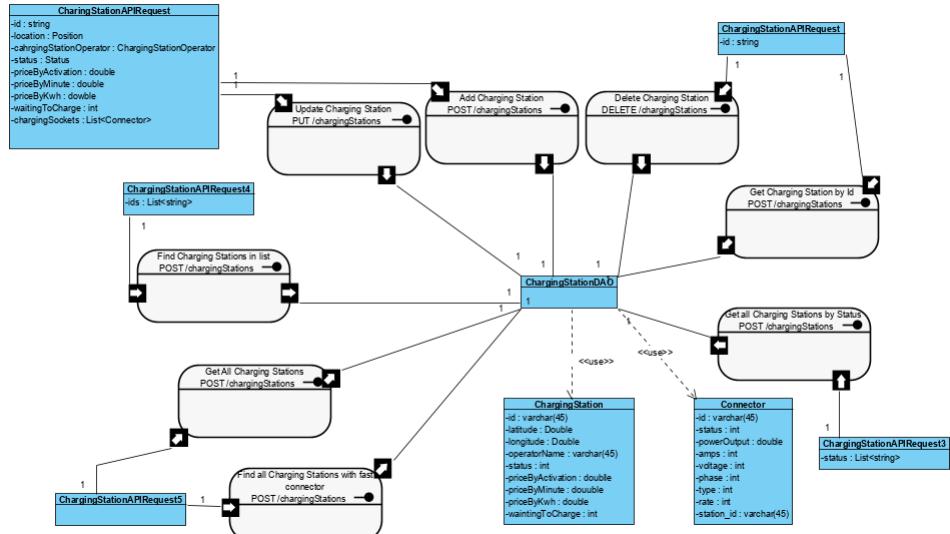


Figura 3.3: Charging Station Calls

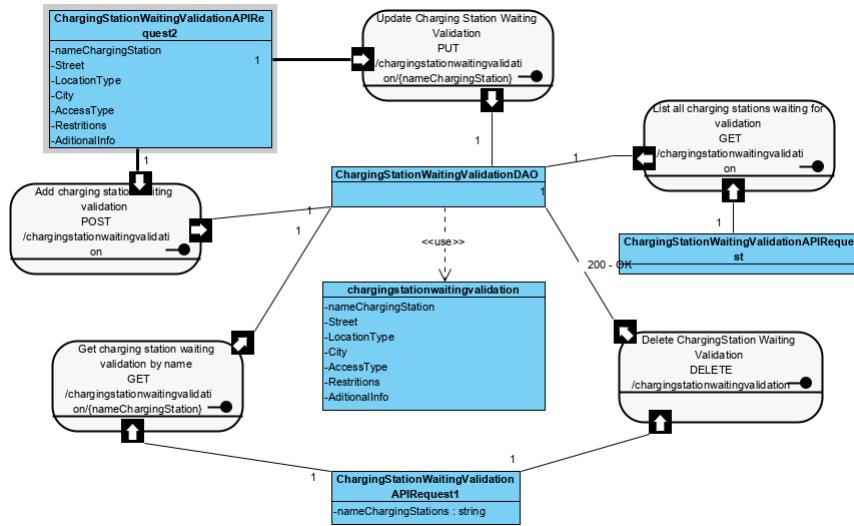


Figura 3.4: Charging Station Without Validation Calls

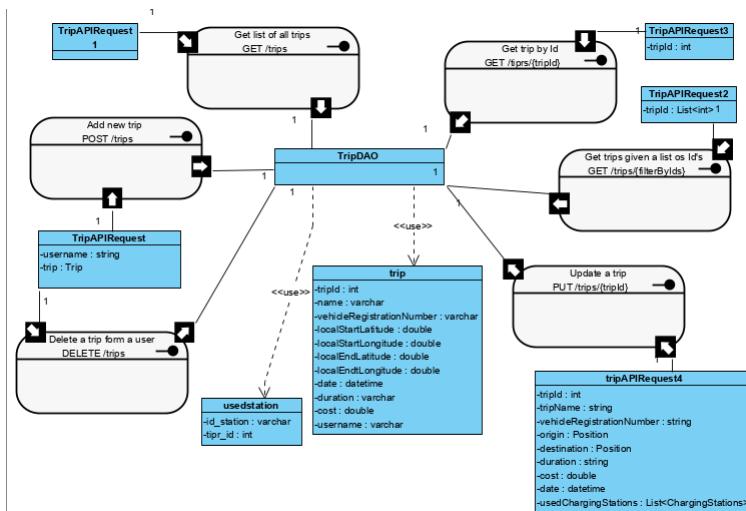


Figura 3.5: Trips Calls

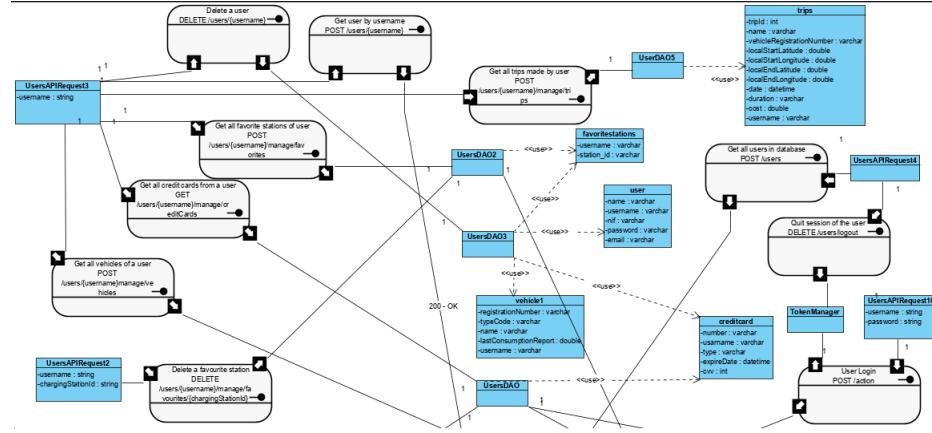


Figura 3.6: Users Calls . 1

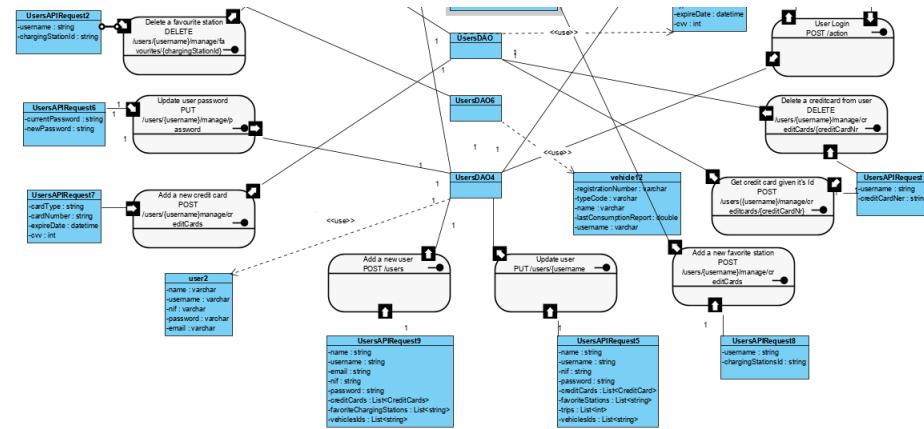


Figura 3.7: Users Calls . 2

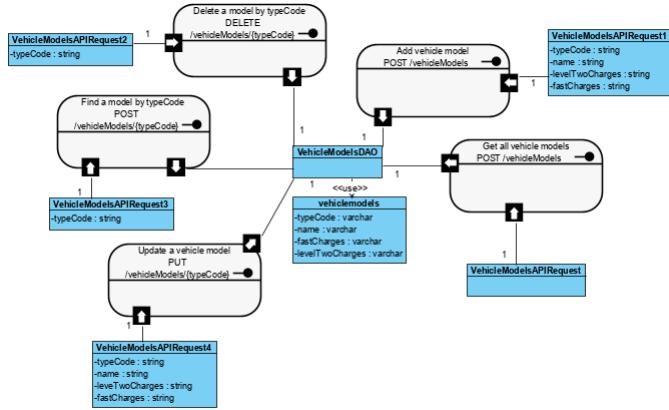


Figura 3.8: Vehicle Models Calls

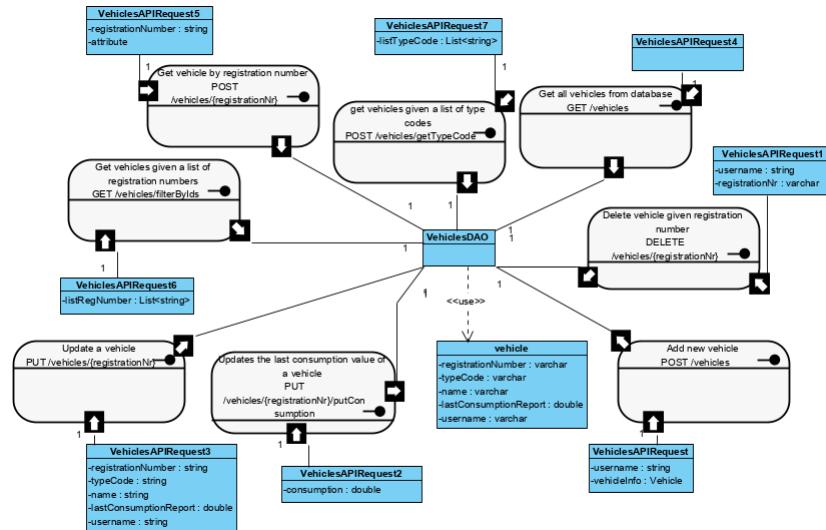


Figura 3.9: Vehicles Calls

Estes diagramas são de leitura e compreensão simples e mostraram-se ao longo do projecto um bom guia para a implementação da nossa aplicação.

3.3 Base de Dados

Para armazenar de modo persistente os dados que consideramos indispensáveis ao bom funcionamento da nossa aplicação, implementamos um Sistema de Gestão de Base de Dados (SGBD) consistente, com o mínimo de redundância dos dados, persistente e seguro, capaz de garantir a integridade e independência dos dados da nossa aplicação. Nessa medida, pretendemos que o nosso SGBD partilhe os dados de uma forma lógica, que permita aos utilizadores inserir, actualizar, remover e recolher dados da base de dados.

Na escolha do modelo de Base de Dados decidimos que a construção de uma Base de Dados Relacional mostra-se adequada para manipular os dados de forma a extrair dos mesmos o conhecimento sobre o planeamento de viagens, os postos de carregamento, etc, assegurando a consistência dos dados e a menor redundância dos mesmos, permitindo a aquisição da informação de forma estruturada e metodológica.

Ademais, consideramos que o investimento numa Base de Dados Relacional será amortizado a médio/longo prazo, visto que este modelo, ao contrário de outros como, por exemplo, um sistema de ficheiros, permite:

- maior eficiência na pesquisa e actualização dos dados;
- suporta transacções, as quais consistem em actualizar um conjunto de dados de forma atómica;
- evita a redundância dos dados, garantindo a robustez e a confiança dos dados, dado que menor redundância é sinónimo de maior integridade dos dados;
- garante a integridade e persistência dos dados, porquanto há coerência e conservação nos dados inseridos na Base de Dados, sendo que estes perduram até serem removidos;
- simplicidade na organização dos dados, visto que os dispõe em tabelas, as quais têm em cada uma das suas colunas os atributos de determinada Entidade, inserindo-se em cada linha uma nova entrada;
- facilidade na apresentação dos dados, possibilitando a consulta e a combinação dos dados de várias tabelas, bem como a filtragem de dados por qualquer atributo, permitindo, desta forma que os dados apresentados sejam somente os relevantes para cada contexto;
- maior flexibilidade. Esta característica é importante quando estão em causa alterações à estrutura da base de dados, como a criação de novas tabelas ou colunas de tabelas existentes. Esta flexibilidade também dá resposta ao número crescente de dados que pode vir a ser introduzido nas tabelas.

Ora, todas estas características da Base de Dados Relacional fazem com que a mesma seja adequada e capaz de armazenar os dados do nosso sistema.

No que se refere ao SGDB utilizado na implementação da nossa Base de Dados, escolhemos o *MySQL* porquanto é um sistema gratuito, o seu código fonte é aberto, a sua API é muito simples, fácil de manipular e corre em diversos sistemas operativos.

Ademais, o *MySQL* é um SGDB seguro e robusto que permite a implementação de regras de segurança no servidor e que permite a utilização de índices para aumentar ainda mais o desempenho.

Finalmente, o facto do *MySQL* seguir os princípios **ACID**, atomicidade, consistência, isolamento e durabilidade, foi ainda um dos motivos que nos levou a adoptar este sistema de gestão e base de dados relacional.

3.3.1 Modelo Lógico da Base de Dados

Na construção da nossa Base de Dados definimos as entidades enumeradas infra e que se mapeiam nas tabelas da nossa Base de Dados:

- ChargingStation;
- ChargingStationWaitingValidation;
- Connector;
- CreditCard;
- FavoriteStations;
- Permissions;
- Trip;
- UsedStations;
- User;
- Vehicle;
- VehicleModels;

As referidas tabelas têm as colunas e os tipos que podemos observar na figura 3.10. Com estas tabelas conseguimos obter toda a informação que necessitamos para desenvolver as funcionalidades da nossa aplicação.

A tabela da *ChargingStation* pretende armazenar os dados referentes aos postos de carregamento existentes no território português. Através desta tabela conseguimos identificar um posto pelo seu id e temos ainda a sua localização, o seu estado, o seu operador, o seu website, os preços praticados e a fila de espera. Por sua vez, a tabela *ChargingStationWaitingValidation* identifica os postos de carregamento introduzidos por um utilizador, mas que ainda aguarda a validação por parte do administrador do sistema. Ainda relativo aos postos de carregamento, a tabela do *Connector* contém todos os atributos das tomadas existentes nos postos de carregamento.

Além destas tabelas, temos ainda a *CreditCard* que tem os elementos que identificam um cartão de crédito. Temos a tabela da *FavoriteStations* que guardam as estações de carregamento preferidas de cada utilizador.

A tabela das *Permissions* identifica o tipo de permissão que um utilizador possui e a tabela *Trip* guarda todos os dados que identificam uma viagem, nomeadamente o número identificador da mesma, o tipo de veículo usado, o local de partida e de chegada, a data, a sua duração, o custo e o utilizador que a efectuou.

Ademais, a tabela *UsedStations* indica os postos de carregamento que foram usados para realizar determinada viagem. Por seu turno, as tabelas *User* e *Vehicle* identificam um utilizador e um veículo eléctrico, respectivamente. De notar que na tabela *Vehicle* foi introduzida a coluna *typeCode* porque a API da Iterino, a qual estamos a utilizar para recolher as características associadas aos diferentes modelos das viaturas, usa esse mesmo *typeCode* para identificar os diferentes veículos e que vai ser indispensável para o planeamento de uma viagem.

Finalmente, temos os *VehicleModels* onde armazenamos os dados referentes a um modelo de automóvel eléctrico. Pela mesma razão apresentada anteriormente inserimos na referida tabela a coluna *typeCode*.

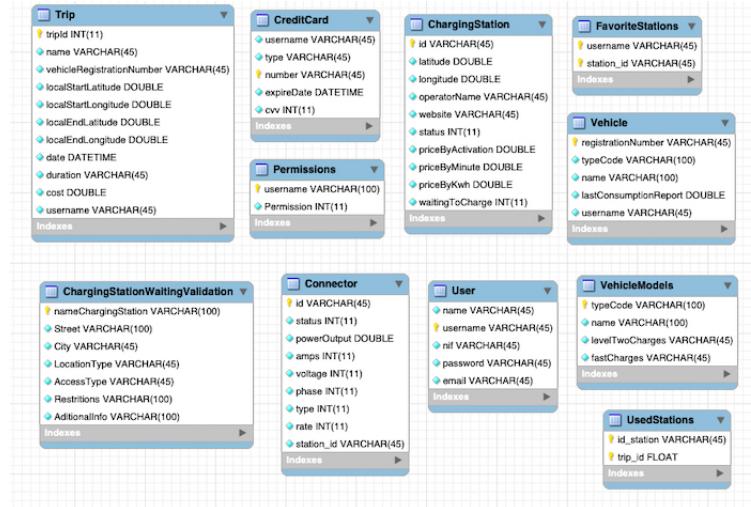


Figura 3.10: Base de Dados - Modelo Lógico

3.3.2 Modelo Físico

De maneira a podermos usufruir de todas as vantagens que o *MySQL* disponibiliza, decidimos utilizar a ferramenta *Workbench* que integra o desenvolvimento, a administração, a criação e a manutenção da Base de Dados num único ambiente de desenvolvimento integrado para o motor de base de dados *MySQL*. Desta feita, a tradução do esquema lógico para o SGBD foi efectuada através do comando *Forward Engineering*, o qual deu origem ao *script* de criação da Base de Dados, que podem ser consultados no Anexo 2.

Capítulo 4

Implementação da Solução Adoptada

O desenvolvimento da aplicação *Plug&GoBeyond* foi dividida em duas parte, o *front-end*, parte da aplicação que interage directamente com o usuário, e o *back-end*, sector responsável, no geral, pela implementação das regras de negócio.

De facto, essa separação está patente na separação física dos programas relativos ao servidor e à parte visual da aplicação que se encontram em pastas separadas, sendo que o servidor tem todo o seu programa implementado na pacote *AppServer* e o *front-end* está codificado no pacote *PGB*.

4.1 Arquitectura

4.1.1 Restful API

O *front-end* e o *back-end* comunicam entre si através de uma *RestAPI*¹ que foi gerada através do *Swagger Editor*², que é um editor *open source* para definir e desenvolver *REST-ful APIs* utilizando a *Swagger Specification*³, ver figura 4.1.

O *swagger* permitiu gerar facilmente uma interface que processa pedidos num formato padronizado, baseado em HTTP *requests*.

Este tipo de API tem grande flexibilidade e portabilidade, permite um maior crescimento horizontal, facilita a escalabilidade, graças à separação entre cliente e servidor e à sua comunicação padronizada. A *Rest API* distinguem quatro tipos de pedidos:

- GET - pedido de algum tipo de dados que se encontram na Base de Dados;
- POST - adicionar uma entrada na Base de Dados;

¹acrónimo de **R**Epresentational **S**tate **T**ransfer

²<https://editor.swagger.io/>

³<https://swagger.io/specification/>

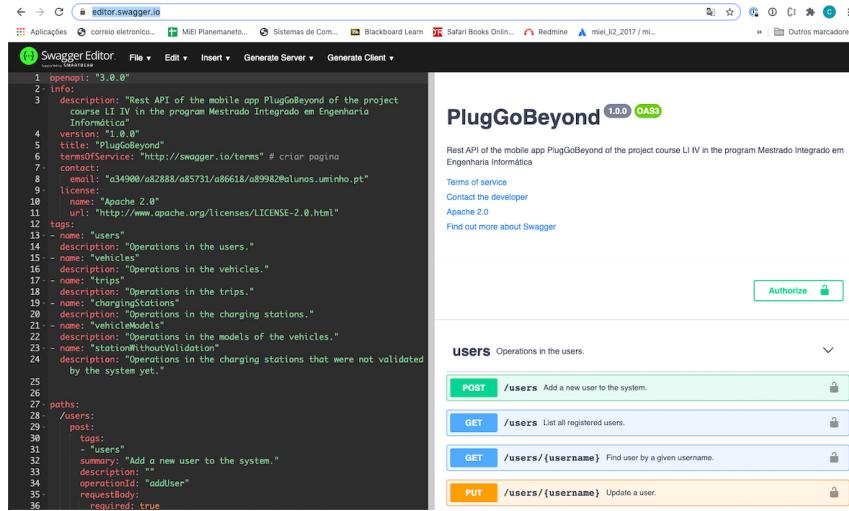


Figura 4.1: Editor do Swagger

- PUT - actualização dos dados da Base de Dados;
- DELETE - apagar dados.

Estes pedidos são feitos ao servidor através de uma URL, que identifica o endereço do mesmo e a operação pretendida. Os referidos pedidos possuem um *header*, que pode conter, entre outros elementos, os dados de autenticação do utilizador, e um *payload* que pode conter detalhes do pedido.

Embora o *payload* do *request* e a resposta possam utilizar outros formatos, de modo geral é utilizado o formato *JSON*, tanto para o envio quanto para o retorno das requisições. Esse formato é escolhido, principalmente, pela sua compatibilidade entre as linguagens e frameworks existentes, tanto no back-end quanto no front-end.

```

1 curl -v -X POST "http://plugandgobeyond.azurewebsites.net/users"
"
2 -H "accept: */*" -H "Content-Type: application/json" -d
3 "{\"name\":\"luis\",\"username\":\"luis\",
4 \"email\":\"luis@gmail.com\",\"nif\":\"123451\",\"password\":
5 \"luis\",\"creditCards\":[{\"cardType\":\"master\",
6 \"cardNumber\":\"123556\",\"expireDate\":\"2020-05-
7 26T15:32:14.497Z\",\"cvv\":120}],\"favoriteChargingStations\":
8 [\"string\"],\"trips\":[0],\"vehiclesIds\"
9 \":[\"string\"]}"
10
11
12 curl -v -X GET "http://plugandgobeyond.azurewebsites.net/users"
13 -H "accept: application/json" -H "Authorization: Bearer
14 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJjZWNpbG1
15 hIiwianRpIjoiOGNiYWIwOTEtZTYwNC00MjhjLTk5MzEtYjJjZmI0MmQ
16 3NDQ3IiwiWF0IjoiMDUvMjcvMjAyMCAtMzo0ODo1MCIsIm5iZiI6MTU

```

```
17      5MDU4NzMcwiZXhwIjoxNTkwNjMwNTMwfQ.mHEN2iIx04gVv8mKu7cg
18      zlcsmQEmqVUnLFel1RHcVXA"
```

No primeiro request estamos a efectuar um POST do utilizador de nome e username luis com os dados que constam do JSON apresentado. Ademais podemos observar que a URL <http://plugandgobeyond.azurewebsites.net/> identifica o endereço do servidor e ainda o caminho *users*, especifica o local onde se encontra o recurso, dentro do servidor.

No segundo exemplo temos um GET de todos os utilizadores da aplicação. Neste *request* podemos verificar que no *header* da mensagem, identificado por *-H*, está contido o *token* de autorização do pedido.

Ademais, em ambos os exemplos o *header* indica ainda que o formato utilizado é JSON.

As calls à nossa API também podem ser feitas através da interface gráfica do *swagger*, conforme o pedido de login e a correspondente resposta que se encontram na figura 4.2.

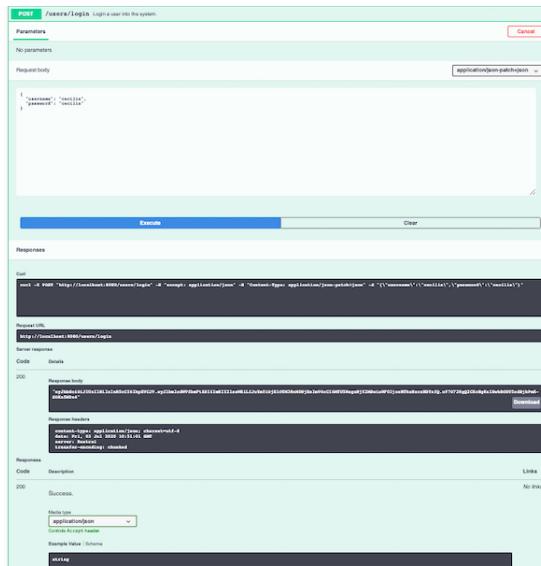


Figura 4.2: Editor do Swagger

4.1.2 Front-End

Integração de APIs Externas

O desenvolvimento da nossa aplicação passou pela integração de diversas APIs externas que nos facilitaram a implementação de diversas funcionalidades. Com efeito, foram utilizadas as APIs para nos auxiliarem na localização dos postos de carregamento e informação sobre as suas tomadas, na identificação das propriedades dos veículos eléctricos e ainda na apresentação dos mapas e rotas de viagem.

Assim, para obtermos a localização dos postos de carregamento utilizamos a API da **Open Charge Map**⁴, para desenarmos os mapas integramos três APIs da **Google Maps SDK for Android**, *Places API* e *Directions API* e, por último, para acedermos às características dos veículos eléctricos, designadamente consumo, modelos, etc., usamos a *Planning API* da **Iterino**.

Modelo Model-View-ViewModel - MVVM

A arquitectura do front-end segue o modelo MVVM, o qual visa estabelecer uma clara separação de responsabilidades, mantendo um *Facade* entre o Model e a View, conforme apresentado na figura 4.3.

Por um lado, a View limita-se a definir a interface que o utilizador vê e o *codebehind* da view, apenas chama o método *InitializeComponent* dentro do construtor e métodos que invoquem novas páginas.

Por seu turno, a ViewModel disponibiliza uma lógica de apresentação, sem ter nenhum conhecimento específico sobre a implementação da View. Esta limita-se a fazer a ponte entre a View e o Model.

De facto, a camada Model não conhece a View, e vice-versa. Na verdade, a View conhece apenas a ViewModel e comunica com a mesma através do processo de *binding*. Este mecanismo faz a ligação entre uma *source* e um *target* consoante a acção a ser executada.

Na nossa aplicação, a interface com o utilizador é escrita em linguagem *Xamarin*, sendo que quando é necessário executar determinada acção utiliza-se o referido mecanismo de *binding* do contexto com o ViewModel e ainda o *binding* de determinado comando ou acção. Entretanto, feita a respectiva ligação com a camada ViewModel, a classe com a qual foi feita essa ligação irá tratar da acção que deve ser executada, fazendo a ponte com a camada Model, caso necessário.

Como exemplo da implementação do modelo supra descrito, apresentamos a rotina do *Login*.

No que se refere à interface com o utilizador, foi utilizada a *framework Xamarin.Forms* e a "Extensible Application Markup Language", XAML. As definições de todas estas interfaces estão no pacote **Pages**.

Em primeiro lugar, temos a interface definida em *Login.xaml*, que faz o *binding* com a classe *LoginViewModel*, conforme o extracto de código que infra apresentamos, fazendo a necessária ligação entre a View e a ViewModel.

⁴<https://openchargemap.org/site/develop>

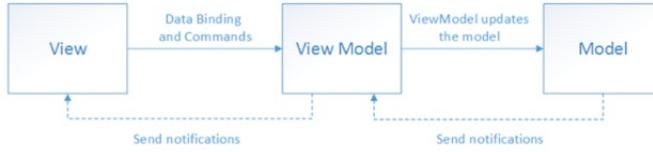


Figura 4.3: modelo MVVM

Acresce que, ainda no mesmo código *xmal*, o botão *Sign In* está associado ao comando *Binding Login*, o qual vai ser tratado na ViewModel, conforme vamos descrever de seguida.

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4      xmlns:d="http://xamarin.com/schemas/2014/forms/design"
5      xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
6      xmlns:viewModels="clr-namespace:PGB.ViewModels;
assembly=PGB"
7      mc:Ignorable="d"
8      Title="Sign In"
9      BackgroundColor="#151723"
10     x:Class="PGB.Pages.LogIn">
11     <ContentPage.BindingContext>
12     <viewModels:LoginViewModel />
13     </ContentPage.BindingContext>
14     (...)
15     <ContentPage.Content>
16         <StackLayout>
17             <Button Text="Sign In"
18                 TextColor="White"
19                 BackgroundColor="#49AF74"
20                 WidthRequest="200"
21                 Margin="20"
22                 HorizontalOptions="Center"
23                 VerticalOptions="Center"
24                 FontFamily="{StaticResource MontserratBold}"
25                 Command="{Binding Login}"/>
26             (...)
27         </StackLayout>
28     </ContentPage.Content>
29 </ContentPage>
```

A classe *LoginViewModel* trata da acção que deve estar associada ao *Login*, definindo o método para o efeito, conforme podemos observar no extracto de código infra. Na verdade, a ViewModel reage a iterações realizadas pelo utilizador. No caso em concreto, quando o utilizador prime o botão do *Sign In* é gerado um alerta ou invocado o método da classe *ApiServices* responsável por enviar os pedidos ao servidor, neste caso, o pedido de *login* com as credenciais do utilizador. Desta forma, é realizada a ponte entre o *ViewModel* e o *Model*, sendo que este último encapsula a lógica de negócios e os dados da aplicação.

```

1 public ICommand Login {
2     get {
3         return new Command(async () => {
4             if (Username.Equals("") && Password.Equals("")) {
5                 await Application.Current.MainPage.DisplayAlert(
6                     "Alert", "Insert an username and password", "OK");
7             }
8             else {
9                 if (Username.Equals("")) {
10                     await Application.Current.
11                         MainPage.DisplayAlert("Alert", "Insert an
username",
12                         "OK");
13                 }
14                 if (Password.Equals("")) {
15                     await Application.Current.MainPage.DisplayAlert(
16                         "Alert", "Insert a password", "OK");
17                 }
18             }
19             if (!Username.Equals("") && !Password.Equals("")) {
20                 var statusCode = await ApiServices.LoginAsync(
21                     Username,
22                     Password);
23                 if (statusCode == HttpStatusCode.BadRequest) {
24                     await Application.Current.MainPage.DisplayAlert(
25                         "Alert", "Invalid username or password.", "OK")
26                 ;
27                 else {
28                     if(statusCode == HttpStatusCode.OK) {
29                         Settings.Settings.Username = Username;
30                         Settings.Settings.Password = Password;
31                         Application.Current.MainPage = new Root();
32                     }
33                 }
34             });
35         }
36     }
}

```

Este modelo permite a utilização de recurso como *databinding*, *commands* e eventos o que torna-o muito eficiente para desenvolver aplicações mais complexas. Ademais, com este modelo torna-se fácil testar e manter a aplicação.

Registrar Utilizador, Viatura e outros dados

O mecanismo para implementar as funcionalidades de registar utilizador, de registar viatura, adicionar viagem, cartão de crédito e adicionar posto favorito são praticamente idênticos, com excepção de que a primeira não necessita que o utilizador tenha uma autorização ou autenticação para registar um novo usuário, ao passo que as outras funcionalidades obrigam a que o cliente tenha uma autorização para fazer cada um dos mencionados pedidos.

Em primeiro lugar, é criado um objecto *HttpClient*, os dados do utilizador ou do recurso que se quer introduzir são passados como parâmetro e, posteriormente, convertidos em formato JSON. De seguida, é criado um *header* do pedido com as informações necessárias e é enviado o *POST request* dos dados, ficando a aguardar pela resposta do servidor ao respectivo pedido.

Na classe *ApiServices* podemos observar todos esta lógica de inserção e de registo de dados. De seguida, apresentamos um pequeno extracto de código que demonstra a implementação da rotina de inserção de dados referida anteriormente:

```

1  public async Task<bool> RegisterUserAsync(User user) {
2      var client = new HttpClient();
3      var json = JsonConvert.SerializeObject(user,
4          Formatting.Indented);
5      HttpContent httpContent = new StringContent(json);
6      httpContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");
7      var response = await client.PostAsync(
8          BaseApiAddress + "/users", httpContent);
9      if (response.IsSuccessStatusCode) {
10          return true;
11      }
12      return false;
13  }
14
15
16
17  public static async Task<bool> RegisterVehicleAsync(string user,
18  Vehicle vehicle,
19  string accessToken) {
20      var client = new HttpClient();
21      client.DefaultRequestHeaders.Authorization = new
22      AuthenticationHeaderValue("Bearer",
23      accessToken);
24      BodyRegisterVehicle body = new BodyRegisterVehicle(user,
25      vehicle);
26      var json = body.ToJson();
27      HttpContent httpContent = new StringContent(json);
28      httpContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");
29      var response = await client.PostAsync(BaseApiAddress +
30      "/vehicles", httpContent);
31      if (response.IsSuccessStatusCode) {
32          return true;
33      }
34      return false;
35  }
36
37  public async Task<bool> AddFavoriteAsync(string user, string id,
38  string accessToken) {
39      var client = new HttpClient();
40      client.DefaultRequestHeaders.Authorization = new
41      AuthenticationHeaderValue("Bearer",
42      accessToken);
43      BodyAddFavorite body = new BodyAddFavorite(id);
44      var json = body.ToJson();
```

```
43     HttpContent httpContent = new StringContent(json);
44     httpContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");
45     var response = await client.PostAsync(
46       BaseApiAddress + "/users/" + user + "/manage/favorites",
47       httpContent);
48     if (response.IsSuccessStatusCode) {
49       return true;
50     }
51     return false;
52 }
```

As suas interfaces do utilizador para o registo e inserção de novos dados têm o design que constam das figuras 4.4 a 4.8.

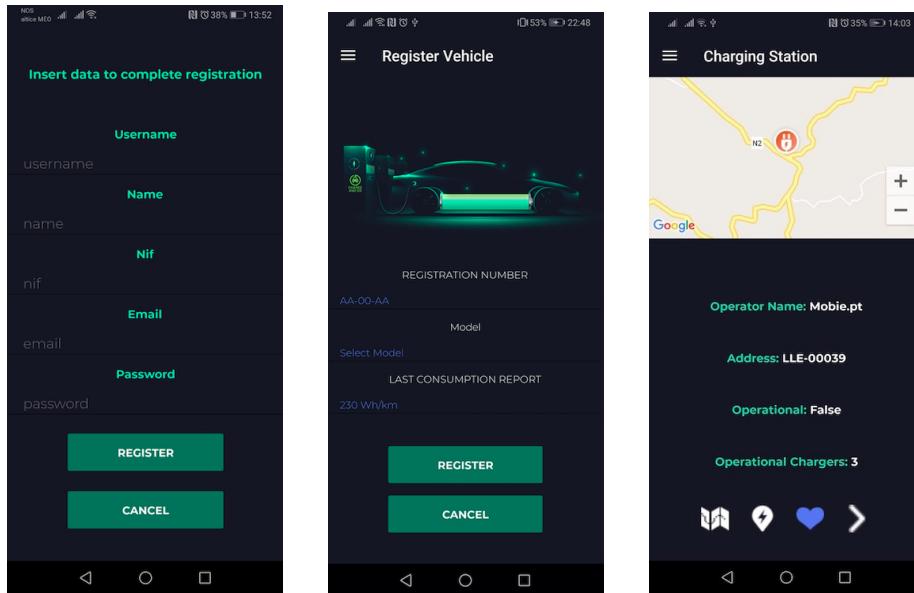


Figura 4.4: Registrar um utilizador

Figura 4.5: Registrar um veículo

Figura 4.6: Adicionar posto favorito

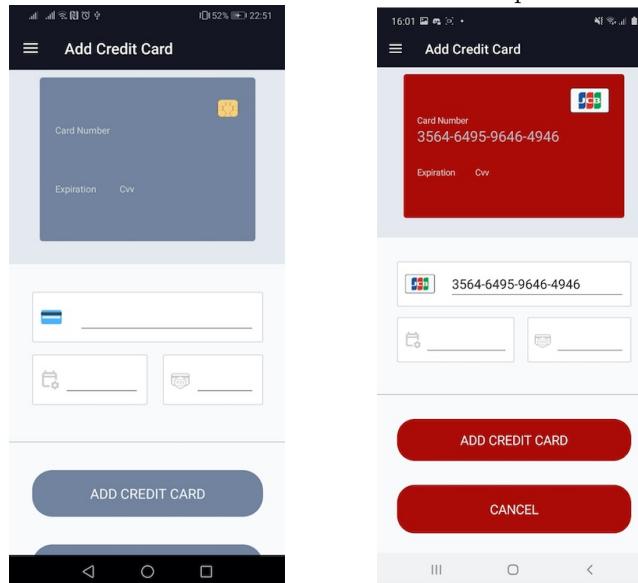


Figura 4.7: Adicionar um cartão de crédito

Figura 4.8: Adicionar um cartão de crédito

Pesquisar Posto de Carregamento

O utilizador tem a possibilidade de pesquisar um posto de carregamento pela localização do mesmo. Para o poder fazer o utilizador tem de entrar na sua área reservada e escolher a opção **Map**, ver figura 4.9. De seguida, aparecer-lhe-á um mapa onde encontra assinalado a localização dos postos de carregamento, conforme mostra a figura 4.10, e basta que na *search bar* da aplicação escreva a localização do posto de carregamento que procura. Depois de inserida a localização, vai ser percorrida a lista de todos os postos de carregamento existentes e verificada se existe algum que contenha a *string* de localização introduzida. A existir apareceram na interface do utilizador todos os postos de carregamento que contêm a localização pesquisada.

Esta lógica está patente no método da classe `SearchStationViewModel` que a seguir transcrevemos.

```
1 public ICommand PerformSearch =>
2     new Command<string>(async(string query) => {
3         Result.Clear();
4         if (ChargingStations.Count > 0) {
5             if (query.Length >= 1) {
6                 foreach (ExternalChargingStationOCM c in
7                     ChargingStations) {
8                     if (c.AddressInfo.Title.ToLower() .
9                         Contains(query.ToLower()) && c.AddressInfo !=
10                         null &&
11                         c.AddressInfo.Title != null && NotInList(c)) {
12                             Result.Add(c);
13                         }
14                     }
15                 });
16             });
17 }
```

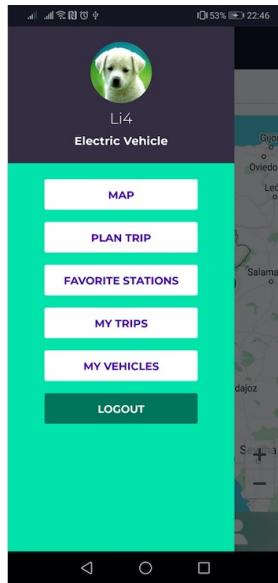


Figura 4.9: Menu das opções



Figura 4.10: Pesquisar posto de carregamento

Permissões para Aceder à Localização do Utilizador

No início da nossa aplicação, no menu principal, é perguntado ao utilizador se permite que a aplicação aceda à localização do dispositivo, conforme figura 4.11. No caso do utilizador recusar nenhuma das funcionalidades da aplicação que necessitam de saber a localização do indivíduo, nomeadamente as funcionalidades de obter direcções para um posto de carregamento ou de planear viagem, ficam disponíveis.

Quando iniciamos a aplicação *PGB.Android* na classe *MainActivity* damos uma implementação específica aos métodos *OnRequestPermissionsResult* e *OnStart*, de modo que quando o usuário inicia a aplicação será verificado pelo sistema se concedeu já permissão para a aplicação aceder à sua localização e, em caso negativo, responde ao pedido de permissão da nossa *app*, sendo que o sistema invoca a implementação do método *onRequestPermissionsResult* por nós definida.

```

1 public override void OnRequestPermissionsResult(int requestCode,
2     string[] permissions, [GeneratedEnum] Android.Content.PM.
3     Permission[] grantResults) {
4     Xamarin.Essentials.Platform.OnRequestPermissionsResult(
5         requestCode, permissions, grantResults);
6
7     Plugin.Permissions.PermissionsImplementation.Current.
8     OnRequestPermissionsResult(requestCode, permissions,
9     grantResults);
10
11 }
```

```

6     base.OnRequestPermissionsResult(requestCode, permissions,
7         grantResults);
8
9     const int RequestLocationId = 0;
10
11    readonly string[] LocationPermissions = {
12        Manifest.Permission.AccessCoarseLocation,
13        Manifest.Permission.AccessFineLocation
14    };
15
16 protected override void OnStart() {
17     base.OnStart();
18     if ((int)Build.VERSION.SdkInt >= 23) {
19         if (CheckSelfPermission(Manifest.Permission.
AccessFineLocation) != Permission.Granted) {
20             RequestPermissions(LocationPermissions,
RequestLocationId);
21         }
22         else {
23             // Permissions already granted - display a message.
24         }
25     }
26 }
```

Ademais, enquanto o utilizador não conceder esta permissão, sempre que este queira fazer uso das funcionalidade que dela necessitam, a mesma será solicitada, conforme figura 4.12, pois não é possível obter direcções para determinado destino a partir da localização do utilizador sem que a aplicação conheça efectivamente a sua localização. Estas solicitações de permissões são operadas através do Xamarin.Essentials antes de invocados os referidos métodos do Android, conforme se pode observar pelo extracto de código que a seguir transcrevemos a título exemplificativo.

```

1 async void PlanTrip(object sender, EventArgs args) {
2     Button button = (Button)sender;
3     Root root = (Root)Application.Current.MainPage;
4
5     var status = await Permissions.CheckStatusAsync<Permissions.
LocationWhenInUse>();
6
7     if (status != PermissionStatus.Granted) {
8         await Permissions.RequestAsync<Permissions.
LocationWhenInUse>();
9         status = await Permissions.CheckStatusAsync<Permissions.
LocationWhenInUse>();
10        if (status == PermissionStatus.Granted) {
11            root.MainMapShowUserLocation();
12        }
13    }
14    else {
15        root.SetDetail("PlanTrip");
16        root.Hide();
17    }
18 }
```

Assim, caso o utilizador queira activar as funcionalidades de obter direcções e planejar uma viagem terá de permitir que o aplicativo tenha acesso à sua localização, conforme podemos observar.



Figura 4.11: Permissão para aceder à localização do utilizador

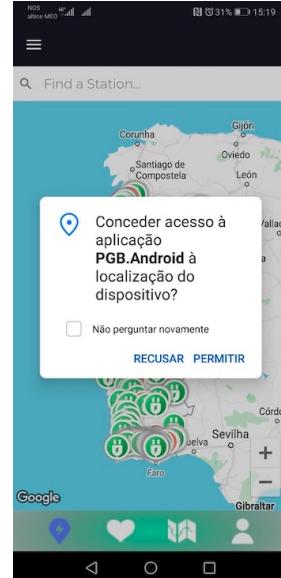


Figura 4.12: Permissão para aceder à localização do utilizador

Planejar Viagem

O caso de planeamento de uma viagem é um caso bastante simples de resolver com um veículo de combustão, onde o tempo de abastecimento do carro é bastante reduzido. No entanto, no caso dos automóveis eléctricos este tema é bastante delicado e o condutor precisa de ajuda para saber qual o melhor trajecto de modo para conseguir abastecer o seu carro eléctrico e chegar ao destino sem problemas.

A nossa aplicação destaca-se bastante neste ponto, resolvendo este problema e dando ao utilizador uma excelente experiência no cálculo de viagens.

Neste cálculo de uma viagem, é essencial saber a capacidade do nosso "tanque de abastecimento", ou seja a capacidade da nossa bateria. Por conseguinte, sempre que o utilizador pretende efectuar uma viagem tem de ter previamente um veículo registado. No registo do veículo é utilizada a API da *Iternio* para o utilizador escolher um modelo (Tesla Model 3, Audi Etron, etc) e a partir deste modelo conseguimos obter um identificador (Typecode) da *Iternio* que nos indica a capacidade da bateria do veículo, consumo médio e outros factores essenciais para determinar uma viagem num carro eléctrico, tais como aerodinâmica, peso, etc.

Após este passo da escolha do veículo, já sabemos qual a capacidade máxima da bateria e como tal é dada a possibilidade ao utilizador de definir uma percentagem inicial na origem, ou seja, o valor percentual relativo à carga da bateria com que vai iniciar a viagem. Este valor é extremamente importante, pois caso não seja tido em conta poderá alterar todo o trajecto e duração de uma viagem. Por exemplo, supondo que temos, na altura do início da viagem, 2% de carga, significa que caso a viagem seja para um local distante, será necessário abastecer primeiro a viatura, o que irá influenciar bastante o tempo da viagem. Este valor caso não seja fornecido é utilizado como referência 100% de carga inicial.

De seguida o utilizador pode ainda decidir dar *override* ao consumo médio estipulado pela *Iternio*. É importante dar a flexibilidade de *override* deste valor, uma vez que caso o utilizador pretenda ir a uma velocidade superior, o consumo médio irá subir drasticamente. Caso nenhum valor seja fornecido é usado o valor como referência da *Iternio*.

Por outro lado, tal como em qualquer sistema de navegação GPS é dada a possibilidade de escolha ao utilizador, se pretende utilizar portagens, autoestradas ou caminhos de ferro para efectuar a viagem. Na nossa aplicação tivemos isso em conta e caso o utilizador seleccione algum destes factores, irá obviamente afectar o tempo total de viagem, uma vez que foram impostas restrições.

Após toda esta informação introduzida, o utilizador basta introduzir uma origem e um destino e iniciar a viagem clicando no botão central. Após iniciada a viagem é ainda dada a possibilidade do utilizador passar todo o trajecto sugerido por nós, com as paragens necessárias, para um navegador GPS à sua escolha (Google Maps, Waze, etc). Nesta fase da viagem iniciada, é dada a possibilidade ao utilizador de ver os detalhes da mesma, clicando no botão para ver detalhes de uma viagem, o utilizador tem acesso ao tempo total da viagem (tempo a conduzir + tempo a carregar), tempo total de carregamento, tempo total a conduzir, distância total até ao destino, número de postos de carregamento pelos quais vai passar, bem como o custo total da viagem, calculado através do **Número de kWh carregados * Preço por kWh**.

Além disso, é apresentada informação relativa à percentagem de bateria na chegada a um posto de carregamento e a percentagem de bateria após carregamento nesse posto, tempo de permanência nesse posto, bem como o tempo de viagem e a distância entre postos de carregamento pelos quais vai passar.

É ainda dada a possibilidade ao utilizador de guardar toda a informação desta viagem na base de dados da Plug&GoBeyond, ver figura 4.16, tendo posteriormente acesso à informação da *trip* realizada, conforme figuras 4.14 e 4.15.

Num contexto mais técnico, esta foi sem dúvida a parte mais complexa da aplicação, pois foi necessário integrar bastantes APIs para conseguirmos apresentar toda a informação ao utilizador da maneira mais *user-friendly* possível.

Inicialmente o utilizador introduz uma origem e um destino, à medida que o utilizador vai escrevendo estão a ser feitas chamadas consecutivas à API da *Google Places* para obter nomes conhecidos para o utilizador seleccionar um. Após escolher um destes nomes, conseguimos imediatamente obter a latitude e longitude do destino e origem introduzidos. Nesta página caso o utilizador introduza a mesma origem e o mesmo destino é apresentada uma mensagem de

erro a informar que a origem e o destino não podem ser os mesmos.

Após este passo, caso o utilizador tente clicar para iniciar a *trip* vai ser apresentada uma mensagem de erro a informar o utilizador que não seleccionou um carro para efectuar a viagem. Após o utilizador entrar nas *settings* para a *trip* é feita uma *call* imediata à API da *Plug&GoBeyond* para obter os veículos desse utilizador. Depois de obtidos os veículos é dada a possibilidade ao utilizador de escolher entre um desses veículos, ou então, caso tenha um veículo como favorito de escolher esse para realizar a viagem. De seguida, aparecem mais uma série de *settings* que já foram referidas previamente (alterar consumo médio, evitar autoestradas, etc).

No passo seguinte, o utilizador clica para iniciar a viagem e é feita uma *API call* à *Iternio* com a origem e o destino introduzidos pelo utilizador e com o *Typecode* (Identificador do veículo) que foram escolhidos para a viagem. A API da *Iternio* devolve uma lista de *Steps*, ou seja passos por onde o utilizador vai ter de passar, com informação útil, como, por exemplo, percentagem de bateria à chegada a um posto, percentagem após carregamento nesse posto etc. Após obtida a informação sobre os postos pelos quais deve passar, é feita uma *API call* à *Google Directions*, com todos os *Waypoints* (Postos de carregamento que temos de passar), bem como a origem e o destino. O resultado da *Google Directions* é uma *encoded string* que representa uma *Polyline*, que é basicamente uma linha a desenhar no mapa entre dois pontos. Para obtermos a Polyline é feito o *decode* da mesma e, após isso, é feito o desenho no mapa da linha respectiva. Por último, são também colocados no mapa todos os postos de carregamento pelos quais o utilizador vai passar, conforme se pode observar na figura 4.13.



Figura 4.13: Percurso de uma viagem

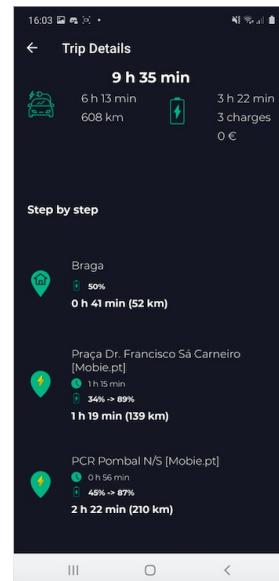


Figura 4.14: Detalhes da viagem

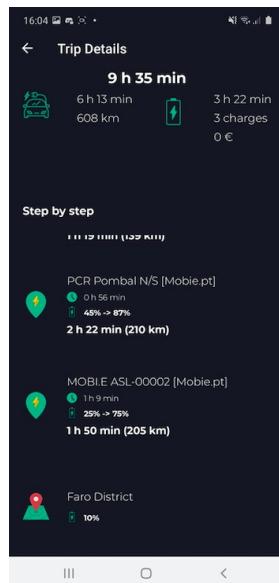


Figura 4.15: Detalhes da viagem

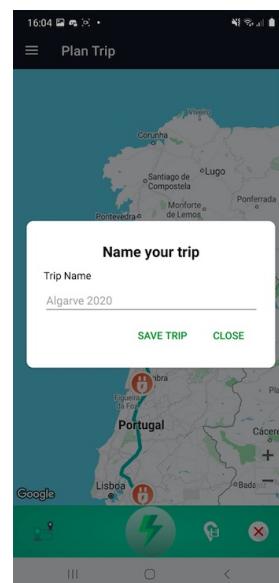


Figura 4.16: Detalhes da viagem

Obter direcções até Posto de Carregamento

Para o nosso utilizador ter a possibilidade de obter direcções para um posto de carregamento a aplicação fornece duas opções: o utilizador pode obter direcções através da página de detalhes ou pode obter direcções para a estação mais perto da sua localização actual, caso este tenha dado permissões de localização à *app*.

Para implementar esta funcionalidade foi utilizada a API da *Xamarin.Essentials*. Na verdade, basta fornecemos as coordenadas do destino e um método de navegação, neste caso é sempre utilizado um carro, e, através da posição do utilizador, a referida API abre uma aplicação nativa como o *Google Maps*, à qual fornece as direcções para uma estação de carregamento.

Caso o utilizador pretenda direcções para uma estação de carregamento específica, através da página de detalhes da estação respectiva, basta carregar na seta que aponta para o lado direito e irá ser aberta uma app nativa com as coordenadas de destino retiradas directamente da página dessa estação.

Caso o utilizador opte pela estação mais próxima, na página do mapa onde contem todas as estações da *Open Charge Map*, ao carregar no primeira imagem no canto inferior esquerdo a app itera sobre a lista de estações e para cada uma calcula a distância euclidiana entre essa estação e a localização do utilizador. Entretanto, encontrada aquela com menor distância abre a app nativa com as coordenadas da estação com a menor distância.

Alterar Password

A nossa aplicação permite que o utilizador altere a sua password, sendo que para isso terá de entrar na sua área reservada e seleccione o ícone que aparece no canto inferior esquerdo do ecrã e o utilizador é redireccionado para uma nova interface onde coloca a palavra antiga, a nova password e repete esta última.

Ademais, convém referir que caso os elementos pretendidos não sejam todos introduzidos ou caso as password introduzidas não estejam correctas são gerados diversos alertas a informar o utilizador que a operação não foi executada e o porquê de ter sido negada a operação.

A implementação deste algoritmo, assim como os demais seguem o modelo MVVM e encontram o seu código nas classes *EditProfile* do pacote Pages, que trata da camada View, e *EditProfileViewModel* do pacote ViewModel.

As interfaces a que fizemos referência são as que estão nas figuras 4.17 a 4.22.

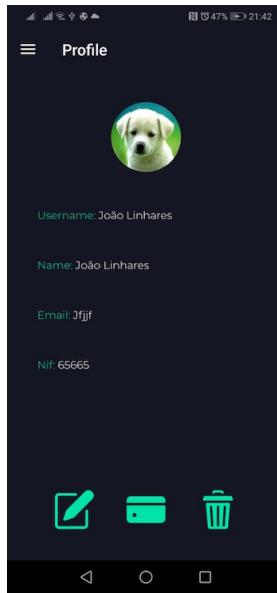


Figura 4.17: Menu de Alterar Password

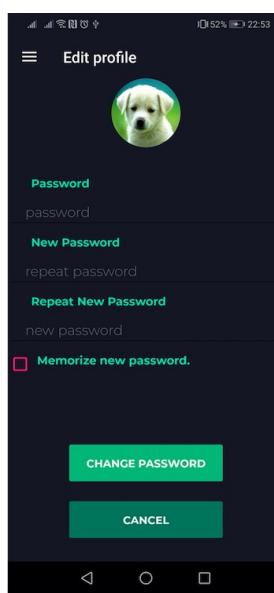


Figura 4.18: Password alterada

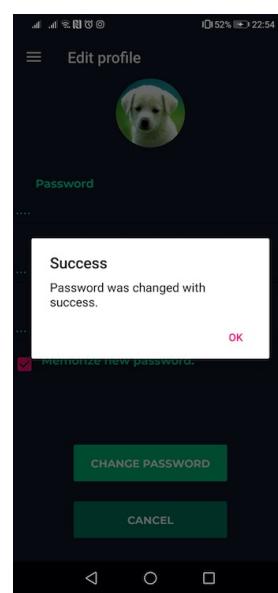


Figura 4.19: Falta confirmar Password

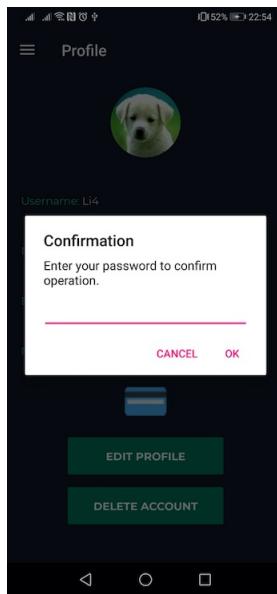


Figura 4.20: Menu de Alterar Password

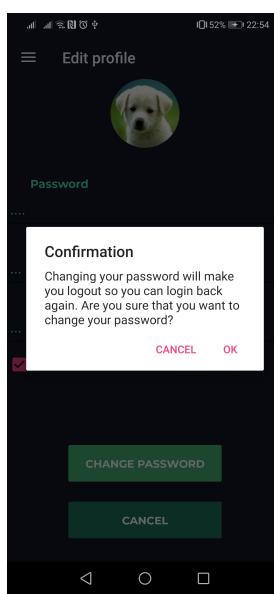


Figura 4.21: Password alterada

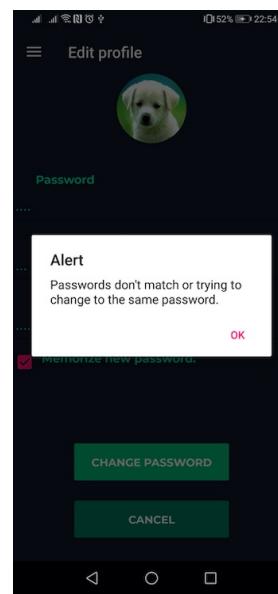


Figura 4.22: Falta confirmar Password

Indicar Viatura e Posto de Carregamento Favoritos

Além das funcionalidades acima explanadas, a *Plug&GoBeyond* permite ao utilizador escolher os postos de carregamento que acha melhores ou mais convenientes, destacando-os como favoritos e guardando na sua área reservada os detalhes dos mesmos.

Ademais, o utilizador tem ainda a possibilidade de estabelecer preferências quanto aos veículos que possui, sendo que para isso terá de escolher um dos seus veículos como favorito. Ao fazer esta selecção, o utilizador está a indicar o veículo que pretende que apareça em destaque no seu perfil, por baixo da sua foto, e quando quiser planear uma viagem, o veículo favorito aparece em lugar de destaque para o caso o utilizador o pretenda escolher para efectuar a viagem.

Estas funcionalidades podem ser observadas nas figuras 4.23 e 4.24.

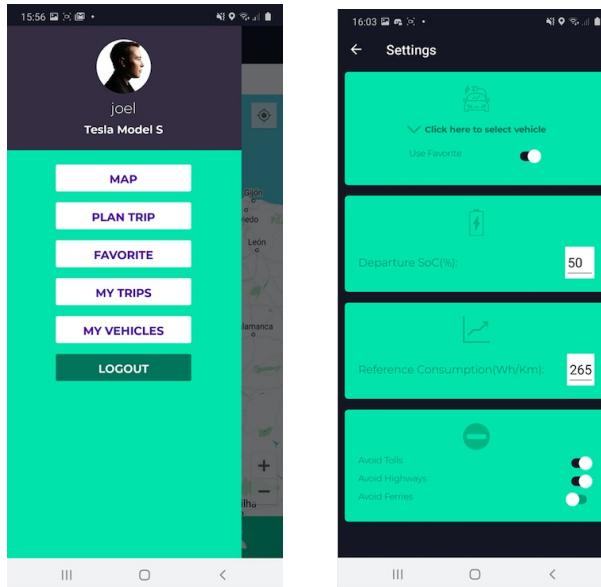


Figura 4.23: Menu Inicial da Área Reservada

Figura 4.24: Settings de Planear Trip

Consultar Dados

Através da nossa aplicação conseguimos fazer uma série de consultas acerca da mais variada informação, designadamente os veículos de um utilizador, os seus postos de carregamento favorito, o seu perfil ou até as viagens que foram efectuadas. Para tal basta seleccionar os menus dos dados que se pretende consultar e a nossa aplicação, pondo em prática o modelo MVVM, disponibiliza as informações correspondentes.

A título exemplificativo, apresentamos as figuras 4.25 e 4.26 que mostram as interfaces que o utilizador vê quando quer obter detalhes dos seus veículos

ou de um veículo em específico.



Figura 4.25: My Vehicles



Figura 4.26: Veículo Favorito

Remover Elementos da Aplicação

Além de adicionar informação, um utilizador pode remover dados relativos ao seu perfil. Um utilizador, depois de aceder à sua área reservada, pode remover determinada informação, deixando assim de ter acesso à mesma. Por outras palavras, depois de um utilizador decidir remover certos dados, os mesmos deixaram de ser visíveis e acessíveis ao utilizador.

Os elementos que o utilizador dizem respeito aos seus veículos, aos postos que identificou com favoritos, aos cartões de crédito que associou a esta aplicação para pagamento dos carregamentos, às viagens que planeou e ainda ao próprio perfil do utilizador.

Estas opções são disponibilizadas em diferentes interfaces, consoante os dados que se pretendem remover.

Através das figuras 4.27, 4.28, 4.29 e 4.30 podemos constatar algumas possibilidades que são dadas ao utilizador no que se refere a remover informação.

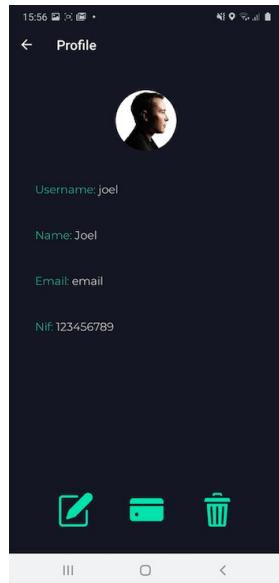


Figura 4.27: Apagar Perfil do utilizador



Figura 4.28: Apagar Posto Favorito



Figura 4.29: Remover veículo



Figura 4.30: Remover viagem

PGBDatabase

Para aumentar a performance e diminuir o *loading time*, bem como as *calls* ao *back-end* ou às APIs externas, decidimos criar uma base de dados local. Para tal, utilizamos o motor de base de dados **SQLite**. Na nossa base de dados local guardamos somente informação relativa aos postos de carregamento e às tomadas dos mesmos. Para isso são criadas duas tabelas, uma para cada uma das entidades referidas.

```

1 static async Task InitializeAsync() {
2 // Responsible for creating the tables if they dont exist
3     if (!initialized){
4         if (!Database.TableMappings.Any(m => m.MappedType.Name == 
5             typeof(ChargingStationDB).Name)){
6             await Database.CreateTablesAsync(CreateFlags.None,
7                 typeof(ChargingStationDB)).ConfigureAwait(false);
8         }
9
10        if (!Database.TableMappings.Any(m => m.MappedType.Name == 
11            typeof(ConnectorDB).Name)){
12            await Database.CreateTablesAsync(CreateFlags.None,
13                typeof(ConnectorDB)).ConfigureAwait(false);
14            initialized = true;
15        }
16    }
17 }
```

Nestas tabelas vamos armazenar as informações acerca dos postos de carregamento, de forma a que quando o utilizador acede à interface *Map*, a aplicação não tem de estar sempre a carregar a informação relativa aos postos de carregamento da API externa *Open Charge Map*, pois esta encontra-se na nossa base de dados local. Isto é possível graças à rotina implementada no *codebehind* da página de xmal designada *Map*, conforme podemos observar do extracto de código que a seguir reproduzimos.

```

1 private async void UpdateMap() {
2     try {
3         // Activate loading Icon
4         IsLoading = true;
5         AddMapStyle();
6         var status = await Permissions.CheckStatusAsync<Permissions
7             .LocationWhenInUse>();
8
9         if (status != PermissionStatus.Granted){
10             MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(new
11                 Position(41.5607359, -8.3984255), Distance.FromKilometers(300))
12             );
13         }
14         else {
15             ShowUserLocation();
16             var loc = await Xamarin.Essentials.Geolocation.
17                 GetLocationAsync();
18
19             MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(new
20                 Position(loc.Latitude, loc.Longitude), Distance.FromKilometers
21                     (300)));
22         }
23     }
24 }
```

```
16
17     }
18     // In case the database is empty
19     var size = App.Database.DatabaseSize();
20     if (App.Database.DatabaseSize() != 834) {
21         await LoadChargingStations();
22     }
23     else {
24         // Get the charging stations from the local database
25         _chargingStations = await App.Database.
26         GetChargingStationsAsync();
27         /*Console.WriteLine("AddToMap\n");*/
28         foreach (var station in _chargingStations) {
29             AddMapPin(station);
30         }
31         await App.Database.SaveChargingStationsAsync(
32         _chargingStations);
33     }
34     catch (Exception ex) {
35         Debug.WriteLine(ex);
36     }
37     IsLoading = false;
38 }
```

4.1.3 Back-End

No *back-end* tratamos do desenvolvimento do nosso servidor, os métodos necessários para responder aos *requests* enviados pelo utilizador e implementamos o padrão de autenticação da nossa *app*.

Ademais, de forma a isolar a camada da lógica de negócio da camada dos dados, também nesta parte da aplicação é gerida a comunicação com a Base de Dados através de *Data Access Objects - DAOs*.

O *back-end* é responsável por recepcionar os pedidos do utilizador, verificar a sua autenticidade, processar o pedido caso o mesmo provenha de um utilizador autorizado, e enviar para o *front-end* a resposta adequada. Esta rotina é conseguida graças às classes contidas no pacote **Controllers**, as quais se encarregam de definir os métodos necessários para, depois de receberem o *request*, aceder à Base de Dados, retirar os dados necessários e processá-los para, finalmente, enviar a resposta.

Controllers

Os métodos presentes no pacote **Controllers**, conforme já referimos, são os responsáveis por fazer a mediação entre os pedidos enviados pelo cliente e o acesso aos dados da nossa Base de Dados através dos DAOs.

As principais classes existentes neste pacote são:

- ChargingStationsApiController;
- TripsApiController;
- UsersApiController;
- VehiclesApiController;

Com estas classes conseguimos desenvolver todos os métodos que permitem ao nosso servidor receber os pedidos do cliente e interagir com a Base de Dados de modo a obter a resposta a esses mesmos pedidos.

A primeira classe encarrega-se de desenvolver os métodos necessários para gerir os dados relativos aos postos de carregamento, nomeadamente inserir um posto de carregamento favorito, encontrar todos os postos de carregamento existentes, encontrar um posto específico por determinado critério, etc.

No que se refere às viagens, a classe *TripsApiController* é responsável por processar toda a informação que a estas dizem respeito, como por exemplo, adicionar uma viagem, encontrar determinada viagem ou todas ou ainda removê-la.

A terceira classe que enumeramos é a que está relacionada com o utilizador e é talvez a que faz a maior carga de trabalho. Com esta classe conseguimos, entre outras acções, registar um utilizador na Base de Dados, iniciar sessão na aplicação e gerar o *token* de autorização para pedidos subsequentes, alterar a *password* de um utilizador, adicionar e remover cartões de crédito, etc.

Finalmente, no que às classes de *controllers* diz respeito, temos a *VehiclesApiController*, esta gera todos os pedidos relativos aos veículos automóveis,

designadamente inserir um carro ou removê-lo. Além destas classes de *controllers*, temos ainda:

- StationWithoutValidationApiController;
- VehicleModelsApiController;

Estas duas classes foram criadas para implementar funcionalidades futuras, pelo que não nos vamos debruçar sobre as mesmas.

No que diz respeito à comunicação entre as várias camadas da aplicação, esta é conseguida graças ao formato padronizado das mensagens trocadas entre cliente-servidor e ainda à rotina de acesso à nossa Base de Dados para inserir e recolher dados.

De seguida demonstramos como é processada a rotina de adicionar um veículo à Base de Dados.

```

1 [HttpGet]
2 [Route("/vehicles")]
3 [Authorize(AuthenticationSchemes =
4 BearerAuthenticationHandler.SchemeName)]
5 [ValidateModelState]
6 [SwaggerOperation("AddVehicle")]
7 public virtual IActionResult AddVehicle([FromBody]VehicleInfo body)
8 {
9     try {
10         VehicleDAO newVehicleDao = new VehicleDAO(DAOController.
11             MyConnection);
12         Vehicle verifyvehicle = newVehicleDao.Get(body._VehicleInfo
13             .RegistrationNumber);
14         Console.WriteLine(verifyvehicle.RegistrationNumber == null)
15         ;
16         if (verifyvehicle.RegistrationNumber == null) {
17             newVehicleDao.Put(body.Username, body._VehicleInfo);
18         }
19         else return StatusCode(400);
20     }
21     catch (Exception) {
22         return StatusCode(400);
23     }
24     return StatusCode(200);
25 }
```

Em primeiro lugar, é obtida uma instância do DAO que acede à nossa Base de Dados e a inspecciona para nos certificarmos de que esse veículo não consta já da mesma.

De seguida, em caso afirmativo, ou seja, o veículo não consta da nossa BD, o mesmo é adicionado e enviada uma mensagem de *Success*, já que o código de status 200 da resposta HTTP indica que o pedido foi concluído com sucesso.

Todavia, se o referido veículo consta já da BD, é enviada a resposta *Invalid parameter*.

Estas mensagens chegam até ao cliente e são apresentadas na UI, conforme podemos observar nas figuras 4.31 e 4.32.

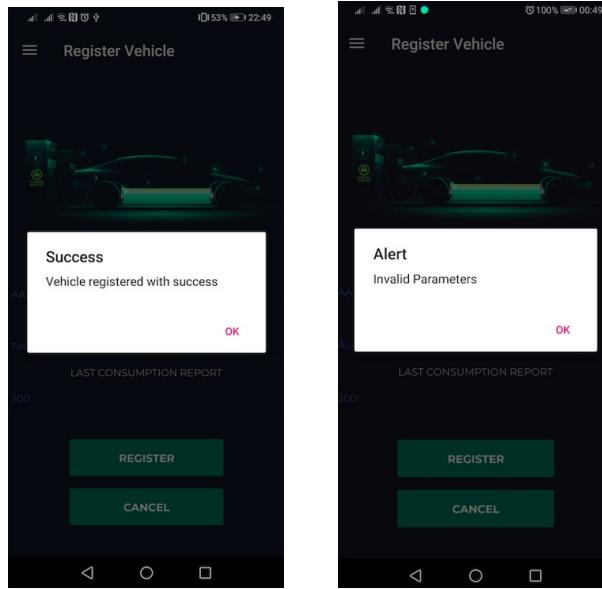


Figura 4.31: Veículo adicionado com sucesso

Figura 4.32: Veículo não registado

Assim, fazendo um resumo da interacção entre as diversas camadas da aplicação, podemos sintetizar este mecanismo da seguinte forma: o cliente envia um pedido ao servidor que começa por verificar se o mesmo está autorizado, em caso afirmativo, é criado um DAO para aceder à Base de Dados e recolher os dados necessários, posteriormente processa-os e envia a resposta para a aplicação cliente.

Autenticação e autorização do Utilizador

Como estamos a desenvolver uma aplicação mobile a questão da segurança não foi descurada, pelo que, por forma a garantirmos a integridade e autenticidade dos nossos dados, decidimos utilizar o mecanismo **JWT (JSON Web Token)**. Este é um método padrão para realizar autenticação entre duas partes por meio de um token.

Um JWT tem uma estrutura tripartida, composta por um *header*, um *payload* e uma *signature*, em que cada uma das suas partes é separadas por um ponto final “.”, conforme podemos observar na figura 4.33.

No que se refere ao *header*, este consiste em duas partes: o tipo de *token*, é o JWT, e o algoritmo de *hashing*.

A segunda parte do *token* é o *payload*, o qual contém os *claims*, que são as informações que queremos armazenar, designadamente a data de expiração do *token*, que no nosso caso são 12 horas. Finalmente, a *signature* é gerada a partir da codificação de uma série de elementos: do *header*, do *payload*, de um

“segredo” e do algoritmo de *hashing* escolhido.

Para utilizarmos o modelo acima descrito e garantirmos que a autenticação e autorização do cliente são implementadas correctamente implementamos duas classes: *BearerAuthenticationHandler*⁵ e *TokensManager*, cujas funções passamos a descrever.

Sempre que o cliente envia um *request* de *login* é gerado um token, através do método *generateToken*, o qual será devolvido ao utilizador para que este faça os restantes *requests* que pretende. Para a criação do *token* tivemos que garantir que era usado um *byteArray* com tamanho suficiente para gerar uma chave simétrica, pelo que tivemos de utilizar uma função de *hash* que dada uma *password*, gera um *byteArray*, com base no algoritmo SHA256, conforme método *generateSecret*. Essa chave simétrica fará parte das credenciais do *JwtSecurityToken* para o caso de ser necessário desencriptar-lo para aceder a informação passada no *payload* mesmo.

Quando o utilizador faz os demais pedidos/*requests*, é enviado no *header* da mensagem o *token*, sendo a autorização dada quando o *token* é validado, através do método *validateToken*, que verifica se o *token* ainda não expirou, se pertence àquele utilizador e se não consta da *BlackList*, lista de *tokens* de utilizadores que terminaram a sessão.

No que diz respeito ao *request* de *logout* em concreto, a nossa estratégia foi criar uma *BlackList* de *tokens* onde armazenamos o nome do utilizador, o *token* e a data de expiração do mesmo. Desta forma, sempre que um utilizador faz *logout* o *token* que estava a ser usado para essa sessão fica armazenado na referida *BlackList*, por meio da função *blackListToken*, e se for utilizado posteriormente não será validado e, consequentemente, negado o *request* que continha o mesmo. Ainda no que se refere a esta *BlackList*, convém mencionar que de tempos a tempos são removidos da mesma todos os *tokens* cujo tempo de validade expirou, de modo a que não sejam armazenados dados desnecessários.

Ademais, convém referir que o user envia o seu *username* e *password* aquando do *login* em *plain text*, pois não estamos a utilizar nenhum protocolo de criptografia, como por exemplo *SSL ou TLS*.

Validamos se o utilizador e a *password* existem na Base de Dados. Entretanto, estamos a armazenar na Base de Dados a *password* e não uma *hash* da mesma.

Em vez de utilizar um *Identity Server* ou servidor de autenticação, implementamos uma autenticação interna para evitar dependências de outras aplicações e de modo a simplificar a nossa implementação.

Ademais, como não usamos um *Identity Server* não implementamos a funcionalidade do *refresh token* porque isso implicaria mais trabalho, pelo que será uma melhoria a efectuar no futuro.

⁵<https://andrewlock.net/a-look-behind-the-jwt-bearer-authentication-middleware-in-asp-net-core/> e <https://stormpath.com/blog/token-authentication-asp-net-core>

The screenshot shows a REST API interface with the following details:

- Responses** section:
 - Curl**: curl -X POST "http://plugandgo.beyond.azurewebsites.net/users/login" -H "accept: text/plain" -H "Content-Type: application/json-patch+json" -d '{"username": "\\"cecelia\\", "password": "\\"cecelia\\""}'
 - Request URL**: http://plugandgo.beyond.azurewebsites.net/users/login
 - Server response** (Code: 200):
 - Response body**: A large JSON token string starting with "eyJhbGciOiJIUzI1Ni...".
 - Download** button.
 - Response headers**:
 - content-encoding: gzip
 - content-type: text/plain; charset=utf-8
 - date: Mon, 19 Jul 2021 17:47:01 GMT
 - server: Microsoft-IIS/10.0
 - transfer-encoding: chunked
 - vary: accept-encoding
 - x-powered-by: ASP.NET
- Responses** section (Code: 200):
 - Description**: Success.
 - Links**: No links

Figura 4.33: Token gerado após pedido de login

Acesso à Base de Dados

O acesso à Base de Dados é feito através de *Data Access Object*, doravante DAO(s), de maneira a que a rotina de acesso aos dados possa ser alterada independentemente do código que utiliza os dados, indo ao encontro de uma arquitectura modular, garantindo maior encapsulamento, facilitando a manutenção do código e a migração da Base de Dados. Estes DAOs utilizam como driver *ADO.NET*⁶, o qual irá estabelecer uma conexão TCP/IP e possibilitar o envio de *requests* à nossa Base de Dados.

Os métodos de criação dos DAOs estão reunidos na pasta *DataAccess*, sendo a conexão à Base de Dados estabelecida a partir da classe **DAOController**. Na figura 4.34. podemos ver em detalhe os DAOs criados.

De seguida, apresentamos um extracto de código relativo ao *VehicleDAO*. O método *infra* adiciona um veículo à Base de Dados. De notar que para garantir a atomicidade das várias operações realizadas introduzimos transacções aquando da inserção de dados na nossa Base de Dados.

```

1 public void Put(String username, Vehicle viat) {
2     MySqlConnection myConn = new MySqlConnection(MyConnection);
3     MySqlTransaction myTrans = null;
4     try {
5         myConn.Open();
6         myTrans = myConn.BeginTransaction();
7         string query = "INSERT INTO Vehicle (registrationNumber,
8             typecode, name, lastConsumptionReport, username) VALUES
9             (@registrationNumber, @typecode, @name,
10             @lastConsumptionReport, @username)";
11         MySqlCommand myCommand = new MySqlCommand(query, myConn,
12             myTrans);
```

⁶ADO.NET é uma "data access technology" da Microsoft .NET Framework

```

13     myCommand.Parameters.AddWithValue("@registrationNumber",
14         viat.RegistrationNumber);
15     myCommand.Parameters.AddWithValue("@typecode",
16         viat.TypeCode);
17     myCommand.Parameters.AddWithValue("@name", viat.Name);
18     myCommand.Parameters.AddWithValue("@lastConsumptionReport",
19         viat.LastConsumptionReport);
20     myCommand.Parameters.AddWithValue("@username", username);
21     myCommand.ExecuteNonQuery();
22     myTrans.Commit();
23     logger.Info("Put vehicle user "+username+" , vehicle:
24     "+viat.RegistrationNumber+" added\n");
25 }
26 catch (Exception e) {
27     try {
28         if (myTrans != null) {
29             myTrans.Rollback();
30         }
31     }
32     catch (MySqlException ex) {
33         if (myTrans.Connection != null) {
34             Console.WriteLine("An exception of type " +
35                 ex.GetType() + " was encountered while
36                 attempting to roll back the transaction.");
37         }
38         logger.Error(ex, " Put vehicle Transaction user "+
39             username + " , vehicle: "+viat.RegistrationNumber+
40             " not rolledback\n");
41     }
42     throw e;
43 }
44 finally {
45     try {
46         myConn.Close();
47     }
48     catch (Exception e) {
49         logger.Error(e, " Put vehicle connection user "+
50             username+ " , vehicle: "+viat.RegistrationNumber +
51             " not closed\n");
52         throw e;
53     }
54 }
55 }
```

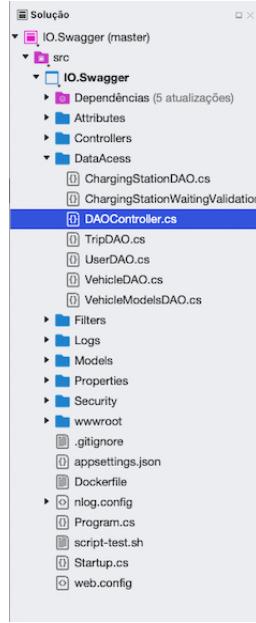


Figura 4.34: Data Access Object

Migração da Base de Dados e migração do *Back-end*

Na fase de desenvolvimento e testes preliminares, tanto o *back-end* como a nossa Base de Dados estavam disponíveis nas nossas máquinas locais. No entanto, visto estarmos a desenvolver uma aplicação com tecnologia predominantemente da *Microsoft*, com excepção da nossa Base de Dados, e cujo *front-end* é uma aplicação mobile, optamos por disponibilizar o nosso produto na **Microsoft Azure**.

Assim o primeiro passo, foi tornar acessível a nossa Base de Dados recorrendo ao serviço *Microsoft Azure MySQL*. A essa base de dados, criada por nós, designamos de **li4pgbdb**.

Ademais, uma das vantagens da utilização deste serviço é a facilidade com que é possível proceder à sua manutenção, actualização e povoamento através da ferramenta MySQL Workbench que utilizamos para desenvolver a nossa Base de Dados.

No que se refere concretamente à integração da nova base de dados no nosso projecto podemos afirmar que a mesma foi extremamente simples. Efectivamente, conforme mencionamos anteriormente, a conexão à base de dados é realizada na classe **DAOController**. Para modificar essa conexão bastou alterar a string **MyConnection**, que agora recebe a nova url do servidor, referente à base de dados alojada no *Microsoft Azure* e as suas respectivas credenciais de acesso.

Num segundo momento, procedeu-se à disponibilização do *back-end* da aplicação

Plug&GoBeyond. Para isso, recorreu-se novamente aos serviços da plataforma *Azure* para disponibilizar o nosso *back-end* na *Cloud*.

Para colocarmos o nosso servidor na plataforma *Azure* tivemos de criar um repositório novo onde guardamos o *back-end*. Posteriormente, utilizamos o serviço de aplicações do *Azure* para criar um grupo de recursos, bem como um plano de serviços de aplicações. Por último, associamos o repositório então criado à mencionada ferramenta⁷.

Esta ferramenta é muito simples de usar e garante total fiabilidade aos programadores, pois permite-nos uma fácil actualização e manutenção do nosso *back-end*, bastando usar o comando *git push azure master* para executar qualquer operação de manutenção ou actualização do *back-end*.

Após este processo de migração, temos o nosso servidor disponível *online* no endereço <http://plugandgobeyond.azurewebsites.net/swagger>, conforme podemos observar na figura 4.35.

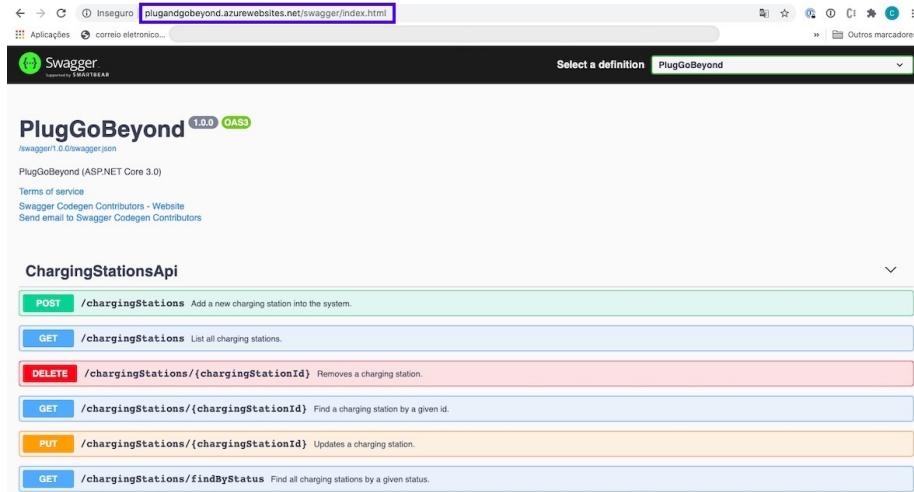


Figura 4.35: URL do servidor na plataforma da Azure

⁷<https://tinyurl.com/y8j3rjqq>

Capítulo 5

Conclusão

A presente aplicação foi desenvolvida com o intuito de trazer maior conforto à comunidade de utilizadores de automóveis eléctricos. Com o presente produto pretendemos oferecer aos utilizadores da mesma uma forma simples, rápida e eficaz de planear as suas viagens, sem se preocuparem com as limitações da bateria do seu automóvel ou com o melhor percurso a seguir. Com efeito, a presente aplicação disponibiliza a funcionalidade de planejar uma viagem, indicando o custo, as paragens, a duração e o percurso da mesma, bastando que o condutor insira apenas a origem e o destino.

Ademais, o presente produto oferece ainda a possibilidade do condutor pedir direcções até a um determinado postos de carregamento ou ainda funcionalidades de carácter mais pessoal, tais como criar uma lista de postos de abastecimento favoritos.

Ora, a reunião destas funcionalidades numa única aplicação garantem-lhe um valor acrescentado em relação às demais existentes no mercado, oferecendo maior comodidade e conforto a quem a utiliza.

Para conseguirmos alcançar o resultado final pretendido a equipa utilizou tecnologia *Microsoft* no *front-end* e *back-end* e uma Base de Dados relacional.

Convém referir que a maior limitação que tivemos, e com a qual nos preparamos todas as semanas, foi a falta de tempo que nos impediu de implementar uma série de *features* que gostaríamos de ter adicionado, mas que serão objecto de trabalhos futuros. Com efeito, temos ainda uma série de ideias e funcionalidades que queremos introduzir, mas como adoptamos um processo de desenvolvimento iterativo e incremental a implementação das mesmas não se encontra comprometida.

Assim, como trabalho futuro, a equipa pretende ainda implementar a possibilidade do condutor iniciar um carregamento remoto; a possibilidade dos utilizadores avaliarem os postos de carregamento segundo critérios predefinidos; permitir a inserção de comentários e fotos acerca de determinado posto de carregamento; permitir que os utilizadores insiram postos de carregamento que ainda não constam da aplicação; permitir que os utilizadores tenham acesso à informação acerca da última vez que determinada tomada de certo posto foi

utilizada e durante quanto tempo, sendo essa informação um indicador se a tomada está operacional, e ainda a possibilidade de se estimar um custo de carregamento.

Por ser um sector em franco desenvolvimento e uma área pouco explorada, consideramos que a nossa aplicação é um bom investimento quer a nível profissional quer a nível financeiro e pretendemos investir nela a médio longo prazo.

Capítulo 6

Anexos

6.1 Anexo 1

Descrição de Use Cases

Na 1:^a fase do nosso trabalho, tendo em conta os cenários apresentados pelo cliente concluímos que o conjunto de acções que o nosso sistema deveria implementar seriam as que de seguida se apresentam.

6.1.1 Use case: Registar utilizador

Descrição: Um novo utilizador pretende-se registar para poder utilizar as funcionalidades da aplicação.

Cenário: O Cristiano, quer registar-se enquanto novo utilizador.

Pré-condição: Ter a aplicação devidamente instalada.

Pós-condição: O utilizador foi adicionado/registado com sucesso ao sistema.

Fluxo normal:

1. O sistema pede que o utilizador seleccione uma operação;
2. Utilizador indica que se pretende registar;
3. Sistema pede para inserir os dados do novo utilizador;
4. Utilizador preenche os dados correspondentes;
5. Sistema verifica validade dos dados inseridos;
6. O sistema adiciona o novo utilizador;
7. O sistema indica que o utilizador foi registado com sucesso.

Fluxo Excepção 1: [Utilizador já se encontra registado] (passo 5)

5.1.Sistema informa que os dados inseridos já pertencem a um utilizador registado;

Fluxo Excepção 2: [Faltam dados por inserir] (passo 5)

5.1.Sistema informa que faltam dados para completar registo;

6.1.2 Use case: Registar viatura

Descrição: Um novo utilizador pretende registar um automóvel.

Cenário: O Cristiano, quer registar o seu novo Tesla como viatura a si associada.

Pré-condição: Estar com sessão iniciada.

Pós-condição: O utilizador registou a viatura com sucesso.

Fluxo normal:

1. O sistema pede que o utilizador seleccione uma operação;
2. Utilizador indica que quer registar nova viatura;
3. Sistema pede para inserir os dados da nova viatura;
4. Utilizador preenche os dados correspondentes;
5. Sistema verifica validade dos dados inseridos;
6. O sistema adiciona a nova viatura;
7. O sistema indica que a viatura registada com sucesso.

Fluxo Excepção 1: [Viatura já se encontra registada] (passo 5)

5.1.Sistema informa que os dados inseridos já pertencem a uma viatura registada;

Fluxo Excepção 2: [Faltam dados por inserir] (passo 5)

5.1.Sistema informa que faltam dados para completar registo;

6.1.3 Use case: Iniciar Sessão

Descrição: O utilizador inicia a sua sessão.

Cenários: O Rui inicia sessão na aplicação.

Pré-condição: O Rui não está já *loggado*.

Pós-condição: O utilizador iniciou a sua sessão no sistema.

Fluxo normal:

1. Utilizador indica quer iniciar sessão como Utilizador registado;
2. Sistema solicita que o utilizador preencha os dados respectivos;
3. Utilizador insere as suas credenciais de acesso;
4. Programa valida acesso;

Fluxo Excepção 1: [Utilizador sem credenciais ou credenciais incorrectas] (passo 4)

4.1.A. Sistema avisa que o utilizador não tem credenciais válidas para aceder ao sistema;

6.1.4 Use case: Terminar sessão

Descrição: O utilizador termina a sua sessão.

Cenário: O Pedro chegou ao seu destino de viagem e quer terminar a sua sessão na aplicação.

Pré-condição: O utilizador do sistema tem sessão iniciada.

Pós-condição: Encerramento da sessão do utilizador.

Fluxo normal:

1. Utilizador solicita o encerramento da sua sessão;
2. O sistema encerra a sessão.

6.1.5 Use case: Adicionar posto de carregamento favorito

Descrição: O utilizador pretende adicionar um posto de carregamento da sua viatura como sendo um dos seus preferidos.

Cenário: O Cristiano quer adicionar o posto Galp Power como um dos seus favoritos.

Pré-condição: O utilizador tem a sessão iniciada.

Pós-condição: O posto de carregamento foi adicionado aos seus postos favoritos.

Fluxo normal:

1. Sistema disponibiliza um mapa com todos os postos existentes no território nacional;
2. Utilizador escolhe o posto que quer adicionar à sua lista de favoritos;
3. Sistema adiciona o novo posto à lista de favoritos do utilizador;

6.1.6 Use case: Planear Viagem

Descrição: O utilizador pretende efectuar uma deslocação de Braga a Faro e quer utilizar a aplicação para saber o melhor trajeto para a viagem, a duração da viagem, tempo de permanência em cada posto, o custo da mesma e qual o tempo de carregamento total, bem como o tempo total a conduzir.

Cenário: O Manuel quer viajar de Braga a Faro e usufruir das funcionalidades da aplicação.

Pré-condição: O utilizador tem a sessão iniciada.

Pós-condição: A informação foi devidamente recebida pelo utilizador.

Fluxo normal:

1. Utilizador indica que quer planejar uma viagem;
2. Sistema pergunta qual o local de origem e de destino da mesma, qual a viatura a ser utilizada, bem como o estado percentual de carregamento da bateria, se quer evitar portagens, caminhos de ferro ou autoestradas e ainda um consumo de referência a utilizar na viagem;
3. Utilizador indica os dados solicitados;
4. Sistema disponibiliza informação acerca da duração da viagem, o custo da mesma, os postos de carregamento que serão utilizados, o tempo de permanência em cada posto e ainda a duração de, eventuais, deslocações entre postos;
5. Sistema dá a possibilidade do utilizador guardar as viagens nas suas trips.

Fluxo Excepção: [Viagem impossível](passo 3)

3.1. Sistema informa que a viagem que o utilizador pretende realizar não é possível com as informações recebidas;

6.1.7 Use case: Seleccionar veículo como favorito numa determinada sessão

Descrição: O utilizador pretende seleccionar um dos seus veículos como favorito na sessão de *login* actual.

Cenário: O Cristiano pretende seleccionar o seu veículo preferido como favorito na sessão actual.

Pré-condição: O utilizador tem a sessão iniciada.

Pós-condição: Veículo seleccionado com favorito com sucesso.

Fluxo normal:

1. Sistema disponibiliza uma lista com todos os veículos do Utilizador;
2. O Utilizador selecciona o veículo que pretende como favorito;
3. O Sistema guarda esse veículo como favorito;

Fluxo Alternativo: [Veículo já está definido como favorito](passo 3)

- 3.1. Sistema informa que veículo já se encontra como favorito;

6.1.8 Use case: Pesquisar Postos

Descrição: O utilizador pretende pesquisar postos através de uma *search bar*.

Cenário: O Cristiano pretende pesquisar postos através de uma *search bar*.

Pré-condição: O utilizador tem a sessão iniciada.

Pós-condição: Lista de postos em que o seu endereço corresponde à pesquisa do Utilizador apresentada com sucesso.

Fluxo normal:

1. Utilizador indica que quer pesquisar postos por nome de endereço;
2. Utilizador insere texto na *search bar* e carrega em procurar;

3. Sistema disponibiliza todos os postos em que o nome do seu endereço contém o texto escrito pelo Utilizador;

6.1.9 Use case: Alterar *Password*

Descrição: O utilizador pretende alterar a sua *password*.

Cenários: O Rui pretende alterar a sua *password*.

Pré-condição: O utilizador está *logado*.

Pós-condição: O utilizador alterou a sua palavra *pass* com sucesso.

Fluxo normal:

1. Utilizador indica quer alterar a sua palavra *pass*;
2. Utilizador insere a sua *password* antiga e duas vezes a nova palavra *pass*;
3. Utilizador selecciona que não pretende guardar nova *password*.
4. Sistema verifica dados.
5. Sistema pergunta a Utilizador se realmente pretende alterar *password*;
6. Utilizador confirma operação;
7. Sistema altera a *password* para a nova.

Fluxo Excepção 1: [Utilizador não inseriu todos os dados] (passo 5)

- 5.1.A. Sistema avisa que o utilizador tem campos por preencher;
- 5.2.A. Volta ao passo 2;

Fluxo Excepção 2: [Utilizador inseriu incorrectamente palavra *pass* antiga] (passo 5)

- 5.1.B. Sistema avisa que o utilizador errou na palavra *pass* antiga;
- 5.2.B. Volta ao passo 2;

Fluxo Excepção 3: [Não há *match* entre a inserção da nova *password* à qual foi inserida duas vezes] (passo 5)

- 5.1.C. Sistema avisa que não há *match* da nova *password*;
- 5.2.C. Volta ao passo 2;

Fluxo Excepção 4: [Utilizador cancela operação] (passo 6)

- 6.1 Volta ao passo 2;

Fluxo Alternativo 1: [Utilizador selecciona para guardar nova *password*] (passo 3)

3.1 Sistema guarda nova *password*;

6.1.10 Use case: Adicionar Cartão de Crédito

Descrição: Um utilizador pretende adicionar um novo cartão de crédito.

Cenário: O Cristiano, quer registar o seu novo cartão de crédito no sistema.

Pré-condição: Estar com sessão iniciada.

Pós-condição: O utilizador registou o novo cartão de crédito com sucesso.

Fluxo normal:

1. Utilizador indica que quer registar um novo cartão de crédito;
2. Sistema pede para inserir os dados do cartão;
3. Utilizador preenche os dados correspondentes;
4. Sistema verifica validade dos dados inseridos;
5. O sistema adiciona a novo cartão de crédito;
6. O sistema indica que o cartão de crédito foi adicionado com sucesso.

Fluxo Excepção 1: [Número de cartão de crédito já se encontra no sistema] (passo 4)

4.1. Sistema informa que os dados inseridos já pertencem a uma viatura registada;

Fluxo Excepção 2: [Faltam dados por inserir] (passo 4)

4.1. Sistema informa que faltam dados para completar registo;

Fluxo Excepção 3: [Dados inválidos] (passo 4)

4.1. Sistema informa que os dados são inválidos;

6.1.11 Use case: Apagar Conta

Descrição: Um utilizador pretende apagar a sua conta.

Cenário: O Cristiano, quer registar apagar a sua conta.

Pré-condição: Estar com sessão iniciada.

Pós-condição: O utilizador removido do sistema com sucesso.

Fluxo normal:

1. Utilizador indica que quer remover a sua conta do sistema;
2. Sistema pede para inserir a *password* actual para confirmar operação;
3. Utilizador insere *password*;
4. Sistema verifica *password*;
5. O sistema remove utilizador com sucesso;

6.1.12 Use case: Obter direcções para a estação mais próxima

Descrição: Um utilizador pretende obter direcções para a estação mais próxima.

Cenário: O Cristiano quer obter direcções para a estação mais próxima para carregar o seu veículo.

Pré-condição: Estar com sessão iniciada e ter dado acesso de localização ao sistema.

Pós-condição: O utilizador obtém direcções para a estação mais próxima.

Fluxo normal:

1. Utilizador indica que quer obter direcções para a estação mais próxima;
2. Sistema calcula estação mais próxima do utilizador;
3. Sistema abre no telemóvel do utilizador numa aplicação nativa com as direcções para a estação mais próxima ;

6.1.13 Use case: Obter direcções para uma determinada estação

Descrição: Um utilizador pretende obter direcções para uma estação à sua escolha.

Cenário: O Cristiano quer obter direcções para uma estação à sua escolha para carregar o seu veículo.

Pré-condição: Estar com sessão iniciada e ter dado acesso de localização ao sistema.

Pós-condição: O utilizador obtém direcções para uma estação à sua escolha.

Fluxo normal:

1. Utilizador escolhe um posto de carregamento;
2. Utilizador indica que quer obter direcções para esse posto;
3. Sistema abre no telemóvel do utilizador numa aplicação nativa com as direcções para a estação mais próxima ;

6.1.14 Use case: Consultar Perfil

Descrição: Um utilizador pretende consultar o seu perfil.

Cenário: O Cristiano quer obter informações sobre os dados da sua conta.

Pré-condição: Estar com sessão iniciada.

Pós-condição: O utilizador obtém as informações da sua conta.

Fluxo normal:

1. Utilizador indica que pretende consultar o seu perfil;
2. Sistema abre uma página à qual o utilizador pode consultar o seu perfil;

6.1.15 Use case: Consultar Viagens Realizadas

Descrição: Um utilizador pretende consultar todas as viagens realizadas.

Cenário: O Cristiano quer obter informações sobre o registo de todas as suas viagens.

Pré-condição: Estar com sessão iniciada.

Pós-condição: O utilizador obtém o registo das suas viagens.

Fluxo normal:

1. Utilizador indica que pretende consultar o registo das suas viagens;
2. Sistema fornece ao utilizador uma lista com todas as viagens realizadas com os seus detalhes;

6.1.16 Use case: Consultar Cartões de Crédito

Descrição: Um utilizador pretende consultar os seus cartões de crédito.

Cenário: O Cristiano quer obter informações sobre os seus cartões de crédito.

Pré-condição: Estar com sessão iniciada.

Pós-condição: O utilizador obtém uma lista com os seus cartões de crédito.

Fluxo normal:

1. Utilizador indica que pretende consultar os seus cartões de crédito;
2. Sistema fornece ao utilizador uma lista com todos os cartões de crédito que pertencem a esse utilizador;

6.1.17 Use case: Consultar Veículos

Descrição: Um utilizador pretende consultar os seus veículos registados.

Cenário: O Cristiano quer obter informações sobre os seus veículos registados no sistema.

Pré-condição: Estar com sessão iniciada.

Pós-condição: O utilizador obtém uma lista com os seus veículos registados.

Fluxo normal:

1. Utilizador indica que pretende consultar os seus veículos registados;
2. Sistema fornece ao utilizador uma lista com todos os veículos registados pelo utilizador;

6.1.18 Use case: Consultar Postos Favoritos

Descrição: Um utilizador pretende consultar os seus postos favoritos.

Cenário: O Cristiano quer obter a lista com os seus postos favoritos.

Pré-condição: Estar com sessão iniciada.

Pós-condição: O utilizador obtém uma lista com os seus postos favoritos.

Fluxo normal:

1. Utilizador indica que pretende consultar os seus postos favoritos;

2. Sistema fornece ao utilizador uma lista com todos os seus postos favoritos;

6.1.19 Use case: Remover Viagem

Descrição: Um utilizador pretende remover uma viagem dos seus registos.

Cenário: O Cristiano quer eliminar uma viagem dos seus registos.

Pré-condição: Estar com sessão iniciada e estar numa página com o registo de todas as suas viagens.

Pós-condição: O utilizador eliminou uma viagem com sucesso dos sistema.

Fluxo normal:

1. Utilizador escolhe qual viagem de uma lista de viagens à qual pretende eliminar do sistema;
2. Sistema pede confirmação de acção;
3. Utilizador confirma decisão;
4. Sistema elimina a viagem dos registos;

Fluxo Excepção 1: [Utilizador cancela decisão] (passo 3)

- 3.1 Volta ao passo 1;

6.1.20 Use case: Remover Cartão de Crédito

Descrição: Um utilizador pretende remover um cartão de crédito do sistema.

Cenário: O Cristiano quer eliminar um dos seus cartões de crédito do sistema.

Pré-condição: Estar com sessão iniciada e estar numa página uma lista de todos os seus cartões de crédito.

Pós-condição: O utilizador eliminou um cartão de crédito com sucesso dos sistema.

Fluxo normal:

1. Utilizador escolhe qual cartão de uma lista dos seus cartões de crédito ao qual pretende eliminar do sistema;
2. Sistema pede confirmação de acção;

3. Utilizador confirma decisão;
4. Sistema elimina o cartão de crédito;

Fluxo Excepção 1: [Utilizador cancela decisão] (passo 3)

- 3.1 Volta ao passo 1;

6.1.21 Use case: Remover Veículo

Descrição: Um utilizador pretende remover um veículo do sistema.

Cenário: O Cristiano quer eliminar um veículo da sua lista de veículos.

Pré-condição: Estar com sessão iniciada e estar numa página com a lista de todos os seus veículos.

Pós-condição: O utilizador eliminou um veículo com sucesso dos sistema.

Fluxo normal:

1. Utilizador escolhe qual veículo de uma lista dos seus veículos ao qual pretende eliminar do sistema;
2. Sistema pede confirmação de acção;
3. Utilizador confirma decisão;
4. Sistema elimina o veículo dos registos;

Fluxo Excepção 1: [Utilizador cancela decisão] (passo 3)

- 3.1 Volta ao passo 1;

6.1.22 Use case: Remover Posto dos Favoritos

Descrição: Um utilizador pretende remover um posto da sua lista de favoritos.

Cenário: O Cristiano quer remover um posto da sua lista de favoritos.

Pré-condição: Estar com sessão iniciada e estar numa página uma lista de todos os postos marcados como favoritos.

Pós-condição: O utilizador retirou um posto da sua lista de favoritos.

Fluxo normal:

1. Utilizador escolhe qual posto da sua lista de favoritos pretende retirar dessa mesma lista;
2. Sistema pede confirmação de acção;
3. Utilizador confirma decisão;
4. Sistema retira posto da lista de favoritos do utilizador;

Fluxo Excepção 1: [Utilizador cancela decisão] (passo 3)

- 3.1 Volta ao passo 1;

6.2 Anexo 2

```

1
2 -- MySQL Workbench Forward Engineering
3
4 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
5 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
   FOREIGN_KEY_CHECKS=0;
6 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY',
   STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,
   ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
7
8 -----
9 -- Schema mydb
10 -----
11 --
12 -- Schema plugandgobeyonddb
13 -----
14
15 -----
16 -- Schema plugandgobeyonddb
17 -----
18 CREATE SCHEMA IF NOT EXISTS `plugandgobeyonddb` DEFAULT CHARACTER
   SET utf8 ;
19 USE `plugandgobeyonddb` ;
20
21 -----
22 -- Table `plugandgobeyonddb`.`ChargingStation`
23 -----
24 DROP TABLE IF EXISTS `plugandgobeyonddb`.`ChargingStation` ;
25
26 CREATE TABLE IF NOT EXISTS `plugandgobeyonddb`.`ChargingStation` (
27   `id` VARCHAR(45) NOT NULL,
28   `latitude` DOUBLE NOT NULL,
29   `longitude` DOUBLE NOT NULL,
30   `operatorName` VARCHAR(45) NOT NULL,
31   `website` VARCHAR(45) NOT NULL,
32   `status` INT(11) NOT NULL,
33   `priceByActivation` DOUBLE NOT NULL,
34   `priceByMinute` DOUBLE NOT NULL,
35   `priceByKwh` DOUBLE NOT NULL,
36   `waitingToCharge` INT(11) NOT NULL,
37   PRIMARY KEY (`id`)
38 ENGINE = InnoDB
39 DEFAULT CHARACTER SET = utf8;
40
41
42 -----
43 -- Table `plugandgobeyonddb`.`ChargingStationWaitingValidation`
44 -----
45 DROP TABLE IF EXISTS `plugandgobeyonddb`.`
   ChargingStationWaitingValidation` ;
46
47 CREATE TABLE IF NOT EXISTS `plugandgobeyonddb`.`
   ChargingStationWaitingValidation` (
48   `nameChargingStation` VARCHAR(100) NOT NULL,
49   `Street` VARCHAR(100) NOT NULL,

```

```

50   'City' VARCHAR(45) NOT NULL,
51   'LocationType' VARCHAR(45) NOT NULL,
52   'AccessType' VARCHAR(45) NOT NULL,
53   'Restrictions' VARCHAR(100) NOT NULL,
54   'AdditionalInfo' VARCHAR(100) NOT NULL,
55   PRIMARY KEY ('nameChargingStation'))
56 ENGINE = InnoDB
57 DEFAULT CHARACTER SET = utf8;
58
59
60 -----
61 -- Table `plugandgobeyonddb`.`Connector`
62 -----
63 DROP TABLE IF EXISTS `plugandgobeyonddb`.`Connector` ;
64
65 CREATE TABLE IF NOT EXISTS `plugandgobeyonddb`.`Connector` (
66   `id` VARCHAR(45) NOT NULL,
67   `status` INT(11) NOT NULL,
68   `powerOutput` DOUBLE NOT NULL,
69   `amps` INT(11) NOT NULL,
70   `voltage` INT(11) NOT NULL,
71   `phase` INT(11) NOT NULL,
72   `type` INT(11) NOT NULL,
73   `rate` INT(11) NOT NULL,
74   `station_id` VARCHAR(45) NOT NULL,
75   PRIMARY KEY (`id`)
76 ENGINE = InnoDB
77 DEFAULT CHARACTER SET = utf8;
78
79
80 -----
81 -- Table `plugandgobeyonddb`.`CreditCard`
82 -----
83 DROP TABLE IF EXISTS `plugandgobeyonddb`.`CreditCard` ;
84
85 CREATE TABLE IF NOT EXISTS `plugandgobeyonddb`.`CreditCard` (
86   `username` VARCHAR(45) NOT NULL,
87   `type` VARCHAR(45) NOT NULL,
88   `number` VARCHAR(45) NOT NULL,
89   `expireDate` DATETIME NOT NULL,
90   `cvv` INT(11) NOT NULL,
91   PRIMARY KEY (`number`)
92 ENGINE = InnoDB
93 DEFAULT CHARACTER SET = utf8;
94
95
96 -----
97 -- Table `plugandgobeyonddb`.`FavoriteStations`
98 -----
99 DROP TABLE IF EXISTS `plugandgobeyonddb`.`FavoriteStations` ;
100
101 CREATE TABLE IF NOT EXISTS `plugandgobeyonddb`.`FavoriteStations` (
102   `username` VARCHAR(45) NOT NULL,
103   `station_id` VARCHAR(45) NOT NULL,
104   PRIMARY KEY (`username`, `station_id`)
105 ENGINE = InnoDB
106 DEFAULT CHARACTER SET = utf8;

```

```

107
108
109 -- Table 'plugandgobeyonddb'.'Permissions'
110 -----
111 DROP TABLE IF EXISTS 'plugandgobeyonddb'.'Permissions' ;
112
113 CREATE TABLE IF NOT EXISTS 'plugandgobeyonddb'.'Permissions' (
114     'username' VARCHAR(100) NOT NULL,
115     'Permissions' INT(11) NOT NULL,
116     PRIMARY KEY ('username'))
117 ENGINE = InnoDB
118 DEFAULT CHARACTER SET = utf8;
119
120
121 -----
122 -- Table 'plugandgobeyonddb'.'Trip'
123 -----
124 DROP TABLE IF EXISTS 'plugandgobeyonddb'.'Trip' ;
125
126 CREATE TABLE IF NOT EXISTS 'plugandgobeyonddb'.'Trip' (
127     'tripId' INT(11) NOT NULL AUTO_INCREMENT,
128     'name' VARCHAR(45) NOT NULL,
129     'vehicleRegistrationNumber' VARCHAR(45) NOT NULL,
130     'localStartLatitude' DOUBLE NOT NULL,
131     'localStartLongitude' DOUBLE NOT NULL,
132     'localEndLatitude' DOUBLE NOT NULL,
133     'localEndLongitude' DOUBLE NOT NULL,
134     'date' DATETIME NOT NULL,
135     'duration' VARCHAR(45) NOT NULL,
136     'cost' DOUBLE NOT NULL,
137     'username' VARCHAR(45) NOT NULL,
138     PRIMARY KEY ('tripId')
139 ENGINE = InnoDB
140 AUTO_INCREMENT = 2
141 DEFAULT CHARACTER SET = utf8;
142
143
144 -----
145 -- Table 'plugandgobeyonddb'.'UsedStations'
146 -----
147 DROP TABLE IF EXISTS 'plugandgobeyonddb'.'UsedStations' ;
148
149 CREATE TABLE IF NOT EXISTS 'plugandgobeyonddb'.'UsedStations' (
150     'id_station' VARCHAR(45) NOT NULL,
151     'trip_id' INT(11) NOT NULL,
152     PRIMARY KEY ('id_station', 'trip_id'))
153 ENGINE = InnoDB
154 DEFAULT CHARACTER SET = utf8;
155
156
157 -----
158 -- Table 'plugandgobeyonddb'.'User'
159 -----
160 DROP TABLE IF EXISTS 'plugandgobeyonddb'.'User' ;
161
162 CREATE TABLE IF NOT EXISTS 'plugandgobeyonddb'.'User' (
163

```

```

164   `name` VARCHAR(45) NOT NULL,
165   `username` VARCHAR(45) NOT NULL,
166   `nif` VARCHAR(45) NOT NULL,
167   `password` VARCHAR(45) NOT NULL,
168   `email` VARCHAR(45) NOT NULL,
169   PRIMARY KEY (`username`)
170 ENGINE = InnoDB
171 DEFAULT CHARACTER SET = utf8;
172
173
174 -----  

175 -- Table `plugandgobeyonddb`.`Vehicle`
176 -----
177 DROP TABLE IF EXISTS `plugandgobeyonddb`.`Vehicle`;
178
179 CREATE TABLE IF NOT EXISTS `plugandgobeyonddb`.`Vehicle` (
180   `registrationNumber` VARCHAR(45) NOT NULL,
181   `typeCode` VARCHAR(100) NOT NULL,
182   `name` VARCHAR(100) NOT NULL,
183   `lastConsumptionReport` DOUBLE NOT NULL,
184   `username` VARCHAR(45) NOT NULL,
185   PRIMARY KEY (`registrationNumber`)
186 ENGINE = InnoDB
187 DEFAULT CHARACTER SET = utf8;
188
189
190 -----  

191 -- Table `plugandgobeyonddb`.`VehicleModels`
192 -----
193 DROP TABLE IF EXISTS `plugandgobeyonddb`.`VehicleModels`;
194
195 CREATE TABLE IF NOT EXISTS `plugandgobeyonddb`.`VehicleModels` (
196   `typeCode` VARCHAR(100) NOT NULL,
197   `name` VARCHAR(100) NOT NULL,
198   `fastCharges` VARCHAR(45) NOT NULL,
199   `levelTwoCharges` VARCHAR(45) NOT NULL,
200   PRIMARY KEY (`typeCode`)
201 ENGINE = InnoDB
202 DEFAULT CHARACTER SET = utf8;
203
204
205 SET SQL_MODE=@OLD_SQL_MODE;
206 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
207 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

```

6.3 Anexo 3

6.3.1 Diagrama de Gantt

1. Brainstorm sobre ideia para aplicação.
2. Definição de use case , diagrama de classes diagrama use cases.
3. Definição das ferramentas a utilizar, e definição de equipas de trabalho.
4. Estruturação do servidor da aplicação.
5. Desenvolvimento do REST API server.
6. Desenvolvimento da aplicação mobile.
7. Implementação do modelo fisico da base de dados.
8. Implementação de métodos de acesso à base de dados.
9. Criação das funcionalidades básicas da aplicação mobile(página de start, iniciar sessão, registar conta, registar veículo e registar posto de carregamento.
10. Geração da API que servirá como servidor.
11. Integração de métodos de acesso à base de dados no Servidor.
12. Integração do acesso da aplicação móvel à API.
13. Inicio da implementação de métodos na API gerada.
14. Configuração das interfaces para as xamls criadas.
15. Inicio da criação do menu principal na aplicação mobile.
16. Implementar o login do utilizador na aplicação mobile.
17. Geração de logs do servidor.
18. Terminar as pages já criadas, mas que se encontram incompletas na aplicação mobile.
19. Reavaliação das calls necessária e refeita a API Server.
20. Correcção na implementação dos logs.
21. Adicionar transações aos métodos de acesso à base de dados.
22. Associar tokens aos utilizadores na autenticação cliente-servidor, de forma a tornar esse acesso mais seguro.

23. Adicionado um ViewModel para cada View para respeitar o modelo MVVM. Cada ViewModel implementa as propriedades e comandos ao qual a View consegue fazer data bind e notifica a mesma caso haja alteração de estado na aplicação mobile.
24. Foi criada a classe ApiService à qual irá conter a definição de todos os métodos necessários para o funcionamento da app que necessitam de comunicação com o servidor.
25. Foram adicionados pontos de carregamento ao mapa.
26. Foram feitas várias melhorias na UI para que a app fosse mais user friendly.
27. Elaboração do relatório final.
28. Estender a autenticação do utilizador ás várias calls.
29. Implementar planeamento das viagens utilizando a API ITERINO.
30. Criar mais métodos na classe ApiService para aumentar funcionalidades da app.
31. Revisão das calls necessárias para a API.
32. Sdicionar funcionalidades ao menu das estações de carregamento favoritas.
33. Selecionar veiculo preferido a partir da lista de veículos.
34. Inserir postos de carregamento na rota da trip.
35. Inserir informação detalhada sobre a trip.
36. Adição de mais funcionalidades de comunicação servidor cliente.
37. Migração do servidor para a plataforma Azure.

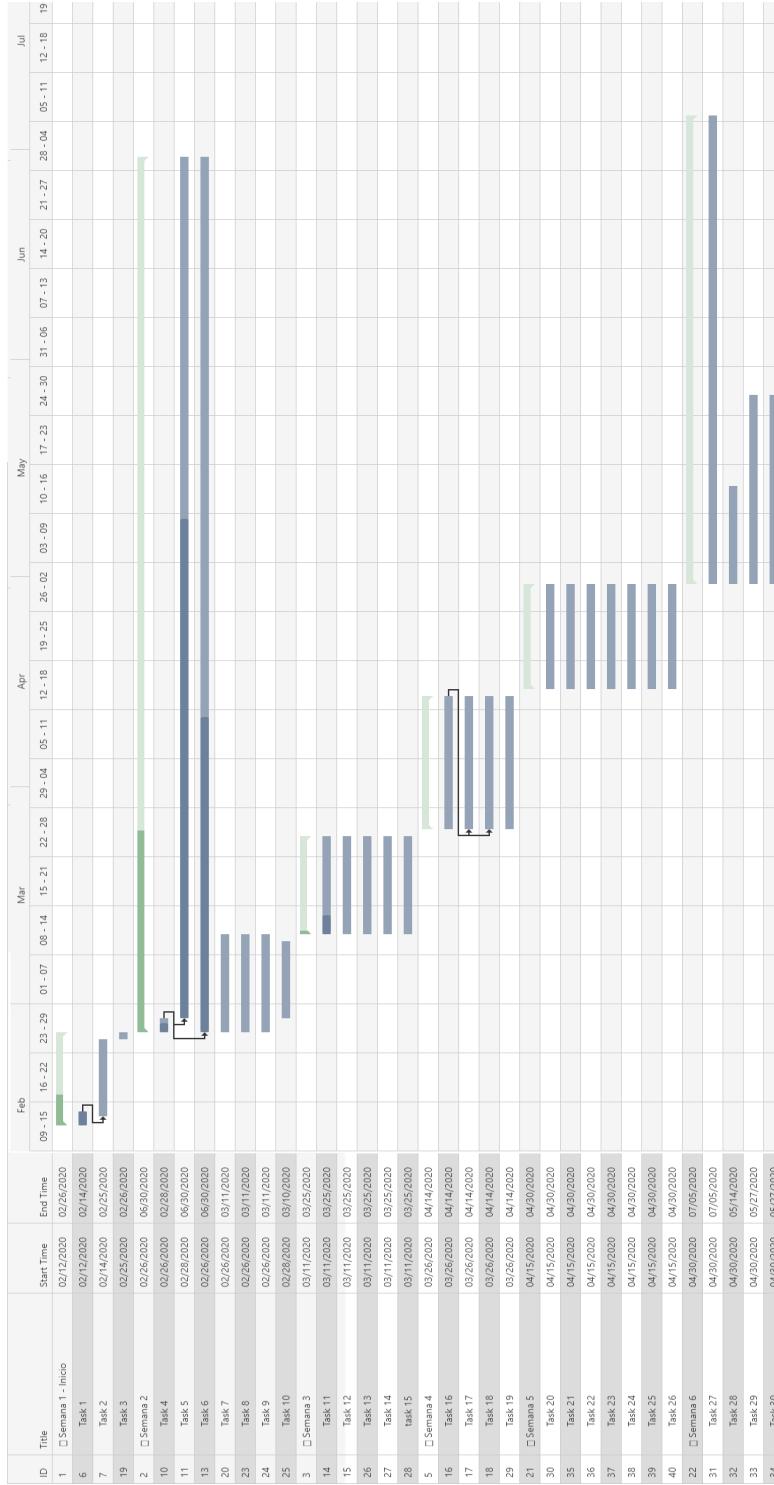


Figura 6.1: Diagrama de Gantt



Figura 6.2: Diagrama de Gantt