



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
DEPARTAMENTO DE ENGENHARIA DE ELECTRÓNICA E
TELECOMUNICAÇÕES E DE COMPUTADORES

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
UNIDADE CURRICULAR DE PROJETO

Dynamic Carpooling System



Autores

Ana Maria Perestrelo Ferreira (45085)

Miguel Diogo Madeira (45083)

Orientador

Professor Doutor Porfírio Pena Filipe

setembro, 2021

Resumo

Com a evolução da humanidade foram criadas invenções essenciais no dia a dia, mas que têm efeitos nocivos para o meio ambiente sendo por isso fundamental procurar alternativas sustentáveis. Os meios de transporte, ao utilizarem principalmente combustíveis fósseis são dos mais poluentes, nomeadamente os transportes particulares.

Para fomentar a sustentabilidade ambiental e diminuir custos, surge o conceito de *Sistema de Boleias* (SB), do inglês *Carpooling System*. Este SB consiste na partilha de lugares disponíveis num veículo particular que realiza uma determinada viagem.

Atualmente estão disponíveis soluções informáticas que se baseiam essencialmente na gestão de anúncios que referem viagens recorrentes ou de longa distância, ou seja, previamente planeadas. Contudo, o *Carpooling*, em zonas urbanas, para ser viável exige flexibilizar a subscrição e negociação de anúncios com importância acrescida em viagens esporádicas e de curta duração.

Neste âmbito, é oportuno conceber e implementar uma solução informática que facilite aos membros de uma comunidade de *Carpooling* o acesso a informação, em qualquer lugar e em tempo real, sobre ofertas de boleias em curso assim como sobre os interessados em aderirem a essas ofertas.

Para o efeito, foi concebida e implementada uma solução informática, no contexto do projeto *Dynamic Carpooling System*, que facilita o acesso e a subscrição de boleias em curso. Esta solução permite que os passageiros visualizem num mapa as boleias que estão a decorrer e que de forma interativa efetuem a respetiva subscrição remetida à aceitação do condutor.

A solução proposta inclui numa interface móvel multi-plataforma, baseada na tecnologia *React Native*, que pode ser executada, por exemplo, em Android e IOS. Esta solução reutiliza uma interface de serviços exportada por um SB de referência.

Palavras-Chave: Sistema de Boleias Dinâmico, Mobilidade Sustentável, Interface de Serviços, Aplicação Móvel.

Abstract

With the evolution of humanity, inventions were created that are essential in everyday life, but that have harmful effects on the environment, making it essential to look for sustainable alternatives. The means of transportation, by using mainly fossil fuels, are one of the most polluting, namely private transport.

To promote environmental sustainability and reduce costs, the concept Carpooling System (SB) arises. This SB consists in sharing available seats on a private vehicle that performs a certain trip.

Currently, there are IT solutions available that are essentially based on the management of advertisements referring to recurrent or long-distance trips, that is, previously planned. However, so that the Carpooling can be viable in urban areas, requires flexible subscription and negotiation of advertisements with increased importance in sporadic and short-term trips.

In this context, it is opportune to design and implement an IT solution that makes it easier for members of a Carpooling community to access information anywhere and in real time, about carpool offers in progress, as well as about those interested in joining these offers.

For this purpose, an IT solution was designed and implemented, in the context of the Dynamic Carpooling System project, which facilitates access and subscription to lifts in progress. This solution allows passengers to view on a map the rides that are taking place and to interactively make the respective subscription, that is sent to the driver to await acceptance.

The proposed solution includes a cross-platform mobile interface, based on React Native technology, which can be run, for example, on Android and IOS. This solution reuses a service interface exported by a referenced Carpooling System.

Key-Words: Dynamic Carpooling System, Sustainable Mobility, Service Interface, Mobile Application.

Agradecimentos

Agradecemos às nossas famílias, que nos apoiaram incondicionalmente durante o nosso percurso académico e que foram muito importantes na nossa evolução como pessoas. Aos nossos amigos por toda a amizade e experiências que proporcionaram ao longo destes anos. Por fim, a todos os docentes que contribuíram para a nossa formação, com os seus conhecimentos e dedicação. E em particular ao nosso orientador.

Índice

Resumo	i
Abstract	iii
Agradecimentos	v
Índice	vii
Lista de Tabelas	ix
Lista de Figuras	xi
Lista de Acrónimos	xiii
1 Introdução	1
1.1 Enquadramento	2
1.2 Motivação	2
1.3 Objetivo	3
1.4 Contribuição	3
1.5 Plataformas Relacionadas	3
1.5.1 Plataformas Privadas	3
1.5.2 Sistema de Boleias de Referência	5
1.6 Organização do Documento	6
2 Tecnologias	7
2.1 Tipos de Aplicação	7
2.1.1 Nativa	7
2.1.2 Multiplataforma	7
2.1.3 Híbrida	8

2.1.4	Comparação	8
2.2	Tecnologias Multiplataforma	9
2.2.1	React Native	9
2.2.2	Flutter	9
2.2.3	Xamarin	9
2.2.4	Comparação	9
2.3	Servidor e Comunicação	10
3	Modelo Proposto e Implementação	13
3.1	Casos de Utilização	13
3.2	Arquitetura	14
3.2.1	Apresentação	15
3.2.2	Lógica	16
3.2.3	Dados	17
3.3	Simulação	22
4	Demonstração	25
4.1	Área Comum	25
4.2	Cenário Passageiro	29
4.3	Cenário Condutor	32
5	Conclusão	37
5.1	Trabalho Futuro	37
5.2	Considerações Finais	37
	Bibliografia	39

Lista de Tabelas

2.1	Tipos de Aplicação - Vantagens e Desvantagens	8
2.2	<i>Frameworks</i> - Vantagens e Desvantagens	10
3.1	<i>Links</i> de Acesso à Máquina Virtual	17
3.2	Efetuar <i>Login</i>	18
3.3	Obter Perfil	19
3.4	Obter Anúncio	20
3.5	Subscrever Anúncio	21
3.6	Aceitar ou Rejeitar Subscrição	22

Lista de Figuras

1.1	Páginas Waze Carpool	4
1.2	Páginas BlaBlaCar	4
1.3	Páginas Scoop Logótipo	5
3.1	Casos de Utilização	14
3.2	Arquitetura DCS	15
4.1	Autenticação	26
4.2	Página Inicial	27
4.3	Perfil - Apenas Passageiro	27
4.4	Perfil - Elegível a Condutor	28
4.5	Ecrã de Passageiro	29
4.6	Viagem Selecionada	30
4.7	Recolha	31
4.8	Resposta à Subscrição	31
4.9	Viagem - Lado do Passageiro	32
4.10	Ecrã de condutor	33
4.11	Viagem - Lado do Condutor	33
4.12	Pedido de Subscrição	34
4.13	Sem Viagem - Condutor	35

Listas de Acrónimos

OMS Organização Mundial de Saúde

SO Sistemas Operativos

API *Application Programming Interface*

CO2 Dióxido de Carbono

REST *Representational State Transfer*

SB *Sistema de Boleias*

DCS *Dynamic Carpooling System*

JSON *JavaScript Object Notation*

KML *Keyhole Markup Language*

HTTP *Hypertext Transfer Protocol*

GPS *Global Positioning System*

Capítulo 1

Introdução

Cada vez mais somos confrontados com as consequências da evolução tecnológica e com o aumento da população o que se refletiu no uso excessivo e impróprio dos recursos naturais que o planeta nos fornece. Atualmente, já não se consegue imaginar o dia a dia sem algumas das invenções que surgiram, nomeadamente o transporte automóvel que será abordado neste projeto.

A grande quantidade de carros nos dias de hoje é a origem de muita da poluição atmosférica existente. Este tipo de poluição traduz-se na presença de substâncias poluidoras no ar atmosférico e representa um problema grave para a saúde da população, dado que pode desencadear problemas cardiovasculares e respiratórios. De acordo com a Organização Mundial de Saúde (OMS) todos os anos morrem sete milhões de pessoas por causas diretamente relacionadas com a poluição [1].

Com as emissões excessivas de gases de efeito de estufa mais especificamente Dióxido de Carbono (CO₂) é possível observar, para além das consequências na saúde, efeitos negativos relacionados com o meio ambiente como o aquecimento global. O aquecimento global define-se como o processo de aumento da temperatura média dos oceanos e da atmosfera, tendo como exemplo de algumas das consequências, alterações na frequência das chuvas, mais secas e ondas de calor e o aumento do nível das águas, motivado pelo descongelamento dos glaciares do ártico [2].

Considera-se fundamental procurar alternativas sustentáveis para reduzir os diversos poluentes emitidos no quotidiano, que ameaçam o planeta e consequentemente a vida dos seus habitantes e das gerações futuras.

1.1 Enquadramento

O conceito de *Carpooling* consiste em partilhar lugares disponíveis num veículo próprio. As viagens estão organizadas, geralmente, em comunidades com interesses comuns e estas pessoas podem eventualmente trabalhar na mesma empresa, morar na mesma localidade, ir ao mesmo evento (cultural, desportivo), entre outros. O *Carpooling* começou a expandir-se nos Estados Unidos da América durante a Segunda Guerra Mundial como forma de preservar os recursos existentes [3].

Como qualquer sistema, o *Carpooling* apresenta vantagens e desvantagens. No que toca às vantagens, este sistema incentiva a sustentabilidade ambiental, reduz as emissões de dióxido de carbono emitidas pelos carros, é também uma forma de poupar dinheiro e conhecer novas pessoas da sua comunidade. Entre as desvantagens está o facto de ainda não ser um sistema muito conhecido e como consequência existem menos viagens disponíveis, não sendo a situação ideal.

Atualmente estão disponíveis soluções informáticas que se baseiam essencialmente na gestão de anúncios que referem viagens recorrentes ou de longa distância, ou seja, previamente planeadas. Contudo, o *Carpooling*, em zonas urbanas, para ser viável exige flexibilizar a subscrição e negociação de anúncios com importância acrescida em viagens esporádicas e de curta duração.

1.2 Motivação

Devido à necessidade e urgência da diminuição dos efeitos das alterações climáticas, é fundamental estudar alternativas que irão garantir a vivência das gerações atuais e futuras. Através do *Carpooling* é possível reduzir os gases com efeito de estufa emitidos pelos carros devido à diminuição de automóveis nas estradas. Não sendo a solução para diminuir toda a poluição, é um passo em frente na sua redução e na consciencialização da população, tendo diversas vantagens económicas e sociais para além da sustentabilidade ambiental.

Em suma, a motivação deste projeto baseou-se em divulgar e promover este sistema de *Carpooling* para diminuir a poluição que os transportes particulares causam.

1.3 Objetivo

O objetivo do projeto *Dynamic Carpooling System* é conceber e implementar uma solução informática que facilite aos membros de uma comunidade de *Carpooling*, no papel de condutores ou passageiros, o acesso, em qualquer lugar e em tempo real, à informação que permite subscrever boleias em curso usando um dispositivo móvel.

1.4 Contribuição

Este projeto procura motivar comunidades de utilizadores de serviços de *Carpooling* através da implementação de um conceito emergente denominado *Dynamic Carpooling*.

Para o efeito, foi concebida e implementada uma solução informática que permite aos passageiros visualizarem num mapa as boleias que estão a decorrer e efetuarem interactivamente a respetiva subscrição remetida à aceitação dos condutores. Esta solução inclui uma interface humana móvel multiplataforma, baseada na tecnologia *React Native*, que pode ser executada, por exemplo, nos sistemas operativos Android ou IOS.

O acesso à informação de comunidades de *Carpooling*, que podem estar alojadas na nuvem, é concretizado adotando a *Application Programming Interface* (API) do sistema de referência descrito na dissertação intitulada Agregador de Sistemas de Boleias [4].

1.5 Plataformas Relacionadas

À semelhança de outros sistemas conhecidos, foram desenvolvidas, ao longo dos anos, várias plataformas que suportam o *Carpooling*. Estas plataformas existentes não suportam o lado dinâmico de requisitar viagens em tempo real, implementado neste projeto.

1.5.1 Plataformas Privadas

Waze Carpool

O Waze [5] é uma aplicação para dispositivos móveis, baseada na navegação por *Global Positioning System* (GPS) e suportada pela Google. Apesar deste

sistema ser mais conhecido como um sistema de navegação, também suporta o Waze Carpool. O Waze está disponível em Portugal e é um auxiliar de condução relativamente conhecido, mas o Waze Carpool ainda não está disponível.

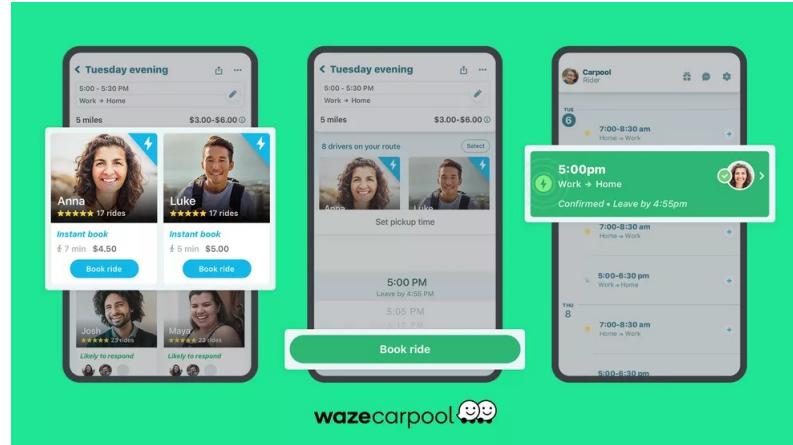


Figura 1.1: Páginas Waze Carpool

BlaBlaCar

BlaBlaCar [6] é uma das maiores plataformas de *Carpooling* existentes a nível mundial e está disponível em vinte e dois países, incluindo Portugal. É descrita como uma rede social que conecta condutores e passageiros e pode ser acedida através da versão Web ou aplicação móvel.

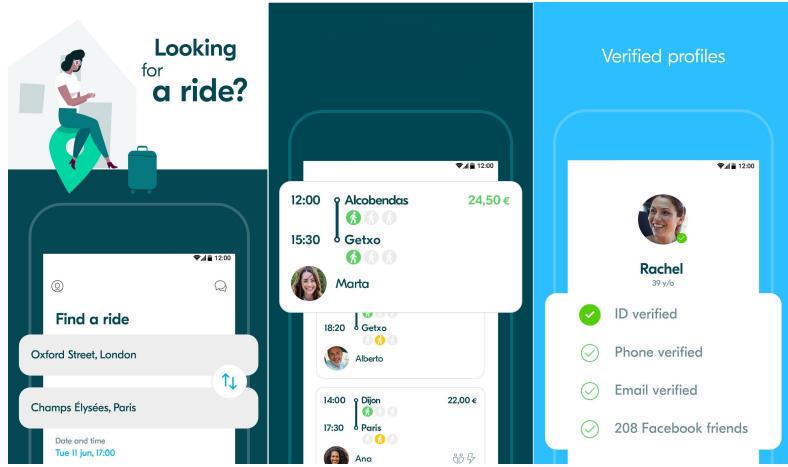


Figura 1.2: Páginas BlaBlaCar

Scoop

Fundado em 2015, o Scoop [7] conecta colegas de trabalho e vizinhos ao fornecer um serviço orientado para comunidades como empresas e habitações.

Este sistema tem como parceiras algumas das maiores empresas mundiais como LinkedIn, FedEx e Samsung, reafirmando a sua segurança. À semelhança do Waze Carpool ainda não está disponível em Portugal.

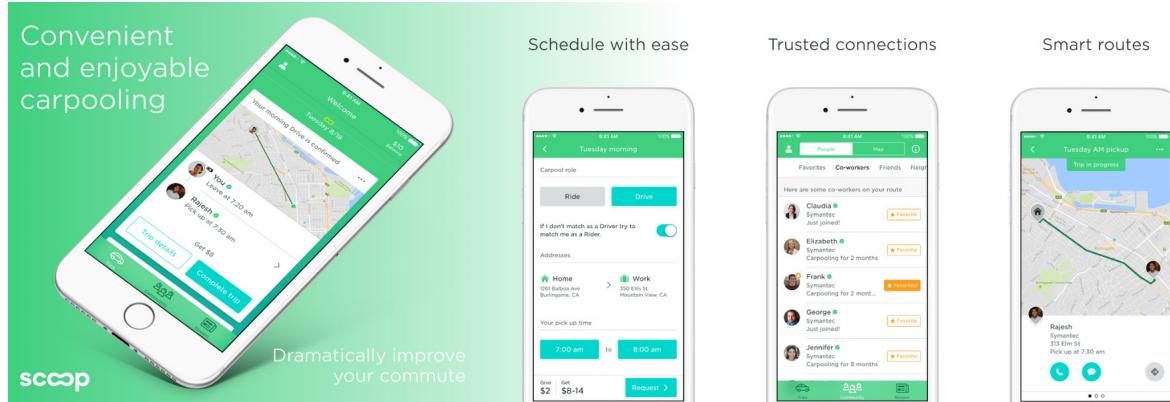


Figura 1.3: Páginas Scoop Logótipo

1.5.2 Sistema de Boleias de Referência

É importante mencionar o Sistema de Boleias (SB) de referência utilizado neste projeto, descrito na dissertação Agregador de Sistemas de Boleias [4], que foi a inspiração impulsionadora para a elaboração do corrente projeto.

Na implementação deste projeto foi feito o acesso à API desenvolvida na camada de serviços do SB de referência, que disponibiliza o acesso aos componentes do SB, “...a camada de serviços concretiza o desacoplamento entre os componentes do SB e as interfaces humanas, atuais ou alvo de desenvolvimento futuro” [4].

1.6 Organização do Documento

Este documento está organizado em seis capítulos. O capítulo atual constitui o primeiro capítulo onde é feita uma introdução ao tema deste projeto, o seu enquadramento, a motivação para o desenvolvimento do mesmo, o objetivo a alcançar, a contribuição e por fim as plataformas existentes relacionadas.

O segundo capítulo efetua uma análise das tecnologias que foram consideradas na implementação deste projeto.

O terceiro capítulo, diz respeito ao modelo proposto, onde são descritos os casos de utilização, é definida a arquitetura a ser implementada e detalhado o processo de implementação.

O quarto capítulo constitui a demonstração, onde está um conjunto de exemplos que mostram as funcionalidades do sistema descritas nos casos de utilização.

O quinto, e último capítulo, conclui o documento com as considerações finais e trabalho futuro.

Capítulo 2

Tecnologias

Dada a natureza da aplicação a desenvolver, é essencial que esta esteja acessível ao maior número possível de utilizadores. Assim sendo, foram analisadas várias tecnologias que suportam o desenvolvimento para dispositivos móveis, dispositivos estes que operam na sua maioria nos Sistemas Operativos (SO) Android e IOS.

2.1 Tipos de Aplicação

Há então a possibilidade de desenvolver três tipos de aplicações [8] direcionadas para os diferentes SO, nomeadamente:

2.1.1 Nativa

Uma aplicação nativa é desenvolvida para um determinado sistema operativo e numa determinada linguagem de programação. Para IOS as aplicações são desenvolvidas em Objective-C ou Swift enquanto para Android são desenvolvidas em Java ou Kotlin.

2.1.2 Multiplataforma

Uma aplicação multiplataforma consiste numa aplicação universal que é suportada pelos diferentes SO, no qual é utilizado apenas um sistema de código. Neste tipo de aplicação há uma experiência de utilizador semelhante à nativa.

2.1.3 Híbrida

Híbrido é algo composto por elementos de tipos diferentes. Uma aplicação híbrida é desenvolvida com a mesma tecnologia utilizada na implementação de *websites* e corre na infraestrutura nativa do dispositivo. É uma combinação da parte nativa e das tecnologias *web* (HTML5, CSS e JavaScript).

2.1.4 Comparação

Devido à necessidade de que este sistema esteja acessível ao maior número de utilizadores possíveis, uma aplicação nativa não foi considerada como opção viável.

Consequentemente restam as aplicações multiplataforma e híbridas, nas quais foram analisadas as vantagens e desvantagens (tabela 2.1). Assim, foi decidido desenvolver uma aplicação multiplataforma por ser menos limitada, uma vez que este projeto requer o acesso a funções do sistema como o GPS dos utilizadores.

	Vantagens	Desvantagens
Nativa	<ul style="list-style-type: none"> • Melhor performance • Maior flexibilidade • Acesso total a APIs nativas 	<ul style="list-style-type: none"> • Código não reutilizável • Não suporta múltiplos SO • Maior custo de desenvolvimento
Multi-plataforma	<ul style="list-style-type: none"> • Suporta vários SO • Interface e experiência semelhante a nativa • Código reutilizável • Desenvolvimento mais rápido e com maior qualidade 	<ul style="list-style-type: none"> • Dificuldade de integração • Pior performance • Acesso mais restrito a funções do sistema (menos limitado que no híbrido)
Híbrida	<ul style="list-style-type: none"> • Suporta vários SO • Manutenção simples • Menos custos de desenvolvimento • Código reutilizável 	<ul style="list-style-type: none"> • Restrição no acesso a funções dos SO • Pior performance • Falta de interação com aplicações nativas

Tabela 2.1: Tipos de Aplicação - Vantagens e Desvantagens

2.2 Tecnologias Multiplataforma

Dentro do desenvolvimento multiplataforma foram encontradas três das *frameworks* mais conhecidas e utilizadas atualmente, sendo estas:

2.2.1 React Native

O React Native [9] é um *framework* Javascript desenvolvido pelo Facebook e é utilizado para desenvolver aplicações para os sistemas Android e IOS, assim como para aplicações *web*. Funciona através de uma “ponte” JavaScript que possibilita a comunicação de JavaScript com a linguagem nativa em questão.

2.2.2 Flutter

O Flutter [10] é um *framework* desenvolvido pelo Google que utiliza a linguagem de programação Dart também criada por esta empresa. Facilita a criação de aplicações multiplataforma com uma interface flexível, mantendo uma performance semelhante à de uma app desenvolvida nativamente.

2.2.3 Xamarin

O Xamarin [11] é um *framework* desenvolvido pelo Microsoft para desenvolver aplicações para os sistemas Android, IOS e Windows. O Xamarin é uma camada de abstração que gera a comunicação de código partilhado com código de plataforma subjacente.

2.2.4 Comparação

Após analisadas as três *frameworks* multiplataforma, através das vantagens e desvantagens descritas na tabela 2.2 [12], a escolha assentou no React Native. Esta *framework* foi a melhor opção entre as estipuladas uma vez que utiliza como linguagem de programação o JavaScript e é uma *framework* muito completa em termos de documentação, sendo complementada por um vasto leque de bibliotecas externas.

	Vantagens	Desvantagens
React Native	<ul style="list-style-type: none"> • Aprender uma vez e escrever em qualquer lado • Javascript no top 5 de linguagens de programação • Documentação extensa • Grande comunidade de programadores 	<ul style="list-style-type: none"> • Performance inferior • Dificuldade no <i>Debugging</i> • Complexa implementação
Flutter	<ul style="list-style-type: none"> • Desenvolvimento mais rápido • Melhor performance • Grande customização 	<ul style="list-style-type: none"> • Recente (menos abrangência) • Pouco utilizado • Linguagem Dart é relativamente recente e requer aprendizagem
Xamarin	<ul style="list-style-type: none"> • Desenvolvimento mais rápido • Performance semelhante à nativa • Manutenção simplificada 	<ul style="list-style-type: none"> • Aplicação mais pesada • Problemas de compatibilidade com bibliotecas externas

Tabela 2.2: *Frameworks* - Vantagens e Desvantagens

2.3 Servidor e Comunicação

Aquando da escolha do React Native como *framework* multiplataforma, as decisões em relação às tecnologias a utilizar no servidor basearam-se na linguagem de programação JavaScript. Ao usar a mesma linguagem ao longo da aplicação o desenvolvimento da mesma torna-se mais eficiente.

O Node.js é uma das tecnologias *backend* mais utilizadas e é uma forma de desenvolver ambientes leves e flexíveis. Esta tecnologia dispõe de uma grande quantidade de pacotes e de documentação. Tendo em conta estas vantagens, optou-se por utilizar o Node.js na implementação do servidor com recurso à *framework* Express. Esta disponibiliza várias ferramentas para o desenvolvimento de aplicações móveis e web [14].

Este projeto ao ter uma componente de comunicação em tempo real, requer

uma biblioteca que suporte a mesma. Para o efeito considerou-se a biblioteca JavaScript Socket.IO [15] que tem na sua essência o uso de WebSockets, todavia disponibiliza serviços que podem ser considerados vantajosos em relação aos WebSockets. Uma das vantagens a salientar é a existência de uma conexão mais fiável que é garantida através do mecanismo de *Hypertext Transfer Protocol (HTTP) Long Polling*, caso haja falha no WebSocket. Tem ainda a capacidade de restabelecer a conexão automaticamente. Por último, permite ainda o uso de *acknowledgements* que pode ser vantajoso caso se pretenda uma implementação de cliente-servidor mais comum, através de pedidos e respostas.

Capítulo 3

Modelo Proposto e Implementação

No modelo proposto é apresentada a solução informática delineada no âmbito deste projeto.

3.1 Casos de Utilização

Os casos de utilização descrevem a forma como o utilizador pode interagir com a plataforma desenvolvida. Neste projeto foram considerados dois tipos de atores, sendo estes, o passageiro e o condutor. A verificação de que um utilizador é condutor é feita através da presença uma carta de condução e carro no sistema.

Na figura 3.1 está representado um diagrama de casos de utilização, que demonstra a forma como o sistema interage com as entidades externas (atores). Ambos os utilizadores podem fazer login e, caso seja feito com sucesso, é autorizado o acesso ao resto das funcionalidades. De seguida, os atores podem ver o seu perfil que é constituído por todas as informações do utilizador. Estas informações podem variar consoante seja passageiro ou condutor. A este último são adicionados os dados sobre a carta de condução e os veículos que este possui.

O passageiro pode ainda visualizar as viagens em curso e selecionar uma delas para subscrever.

O condutor pode iniciar uma viagem programada, terminar a viagem que está a realizar e aceitar ou recusar uma subscrição feita pelo passageiro.

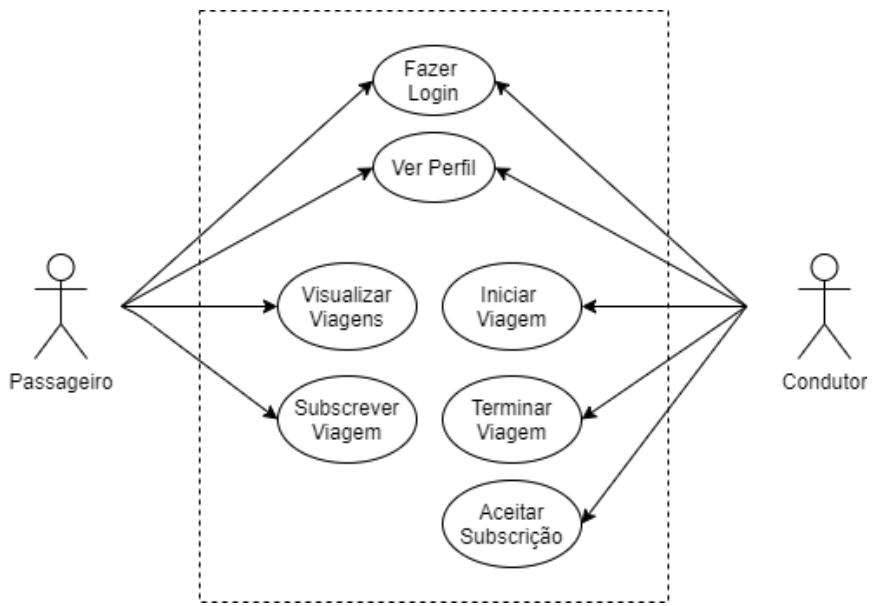


Figura 3.1: Casos de Utilização

3.2 Arquitetura

Um dos passos mais importantes, senão o mais importante, no desenvolvimento de um projeto é a conceção de uma arquitetura que irá ser utilizada na fase de implementação.

Através desta arquitetura é possível ter uma visão geral das diferentes componentes necessárias na solução informática, que na sua totalidade descrevem o *Dynamic Carpooling System* (DCS) desenvolvido neste projeto.

Como se pode observar na figura 3.2 o DCS encontra-se dividido em três camadas sendo estas, a camada de apresentação, a camada de lógica e a camada de dados.

Os dispositivos móveis, representados por um telemóvel e por um *tablet*, retratam a camada de apresentação. É nesta camada que estão implementados os componentes da interface de utilizador.

A camada de lógica encapsula os serviços necessários aos utilizadores, quer sejam implementados de origem ou provenientes da API de serviços utilizada.

Por fim, existe a camada de dados que é composta pela API exportada pelo sistema de referência utilizado na dissertação Agregador de Sistema de Boleias [4]. Esta camada é responsável por fornecer o acesso aos dados armazenados no SB.

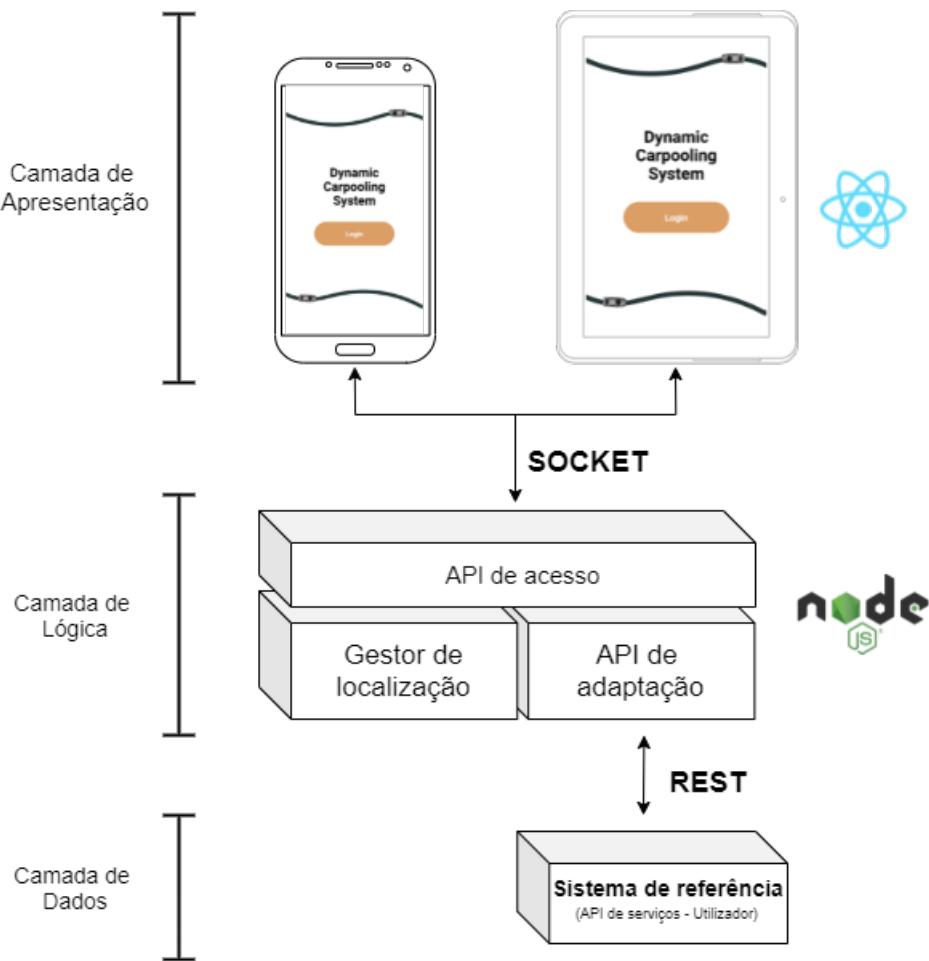


Figura 3.2: Arquitetura DCS

3.2.1 Apresentação

A camada de apresentação é o meio de comunicação entre o sistema e o utilizador, quer este seja condutor ou passageiro. Como referido nos casos de utilização, há um conjunto de funções comuns ao condutor e ao passageiro e funções específicas para cada tipo de utilizador.

Na área comum é possível efetuar login e caso seja feito com sucesso, reencaminha o utilizador para a página inicial. Nesta página o utilizador poderá sempre escolher visualizar o seu perfil e, caso seja elegível como condutor, poderá escolher entre aceder ao ecrã de passageiro ou de condutor. Caso seja ilegível, apenas é disponibilizado o acesso ao ecrã de passageiro. Um utilizador elegível a condutor possui uma carta de condução e um veículo.

3.2.2 Lógica

Ao descrever a camada de lógica é imprescindível compreender alguns conceitos base, como por exemplo a definição de cliente e servidor. Um cliente é a representação de um utilizador por via do dispositivo em que se encontra e tem a capacidade de receber informação ou utilizar um serviço particular dos servidores. Um servidor pode ser um programa de computador ou um dispositivo que gere informação ou fornece serviços a clientes.

Ao implementar um sistema como o DCS é necessário garantir a comunicação bidirecional entre servidor e cliente para a troca de localizações entre clientes. Uma comunicação bidirecional baseia-se no conceito de que tanto o servidor como o cliente podem enviar informações entre si. Deste modo, o cliente não precisa estar sempre a efetuar pedidos de atualizações, sendo informado pelo servidor quando estas ocorrem.

A camada de lógica corresponde a um servidor desenvolvido com recurso à *framework* Express, que provém do Node.js. As comunicações efetuadas entre o servidor e os clientes foram implementadas através do Socket.IO.

Esta camada é a camada chave na interação entre os diferentes utilizadores uma vez que é o “meio de transição”. Encontra-se dividida em três componentes: a API de acesso, o gestor de localização e a API de adaptação.

A API de acesso engloba todas as funcionalidades relacionadas com obtenção de dados da API de serviços do sistema de referência, demonstrado na figura 3.1 e o gestor de localização em tempo real.

A gestão das localizações dos diferentes utilizadores é um ponto essencial deste sistema. O gestor de localização é responsável por receber a informação relativa à localização de ambos os tipos de utilizador, sejam estes passageiro ou condutor. O condutor, enquanto realiza uma viagem, envia a sua localização para o servidor através de um *socket*, onde fica guardada. Caso hajam passageiros à procura de viagens nas proximidades do condutor, a localização do condutor é enviada da mesma forma e apresentada no ecrã do passageiro. O passageiro apenas envia a sua localização quando efetua um pedido de boleia. Após o pedido ser processado, as coordenadas relativas ao passageiro são enviadas diretamente para o condutor que está a realizar a viagem.

A API de adaptação responsabiliza-se por fazer pedidos à API de serviços do sistema de referência.

3.2.3 Dados

Como é possível observar na figura 3.2, a camada de dados neste projeto não foi implementada, mas foi feito um acesso a um sistema referência de boleias existente. Foram consideradas diferentes opções relativamente à arquitetura a desenvolver, nomeadamente a implementação de um SB ou a reutilização de um SB. No entanto não se optou pela hipótese de implementar uma camada de dados de raiz, uma vez que várias empresas já desenvolveram este tipo de sistema. Assim sendo, procedeu-se à reutilização de um SB existente.

Ao reaproveitar um SB existente é possível utilizar na camada de dados qualquer sistema de referência com uma estrutura semelhante ao utilizado.

A arquitetura de referência do SB utilizado possui uma camada de serviços onde está alojada uma API. Esta API foi implementada com recurso a serviços *Representational State Transfer* (REST), tornando-se assim numa RESTful API. O acesso à informação é feito através de pedidos REST e entregue via HTTP com recurso a *JavaScript Object Notation* (JSON). O formato JSON é um dos formatos mais utilizados uma vez que pode ser lido por máquinas e humanos.

Todos os pedidos efetuados à API do sistema de referência neste projeto são realizados com recurso a parâmetros e devolvem um estado numérico e uma resposta em formato JSON. Se o estado devolvido for '200', o pedido REST foi concluído com sucesso. Senão, consoante o erro, é devolvido o aviso correspondente.

Na tabela 3.1, na secção referente à interface de serviços, estão descritos os *links* utilizados no acesso à API, referentes aos anúncios e ao utilizador.

Interface Gráfica:	http://40.68.190.183:8091/CSS/index
Interface de Serviços:	
Anúncios	http://40.68.190.183:8091/CSSRest/announcement/
Utilizador	http://40.68.190.183:8091/CSSRest/user/

Tabela 3.1: *Links* de Acesso à Máquina Virtual

Ao iniciar a aplicação móvel o utilizador é reencaminhado para o *login*. Após inserir as suas credenciais e pressionar o botão de *login* é utilizado o método POST para realizar a autenticação das mesmas. Na tabela 3.2 é possível observar os parâmetros enviados (*email* e *password*) e caso haja a validação destas credenciais é devolvido o identificador do utilizador.

O identificador do utilizador, aquando da sua validação, é armazenado no *AsyncStorage* do React Native. O *AsyncStorage* possui um formato chave-valor e permite armazenar dados globalmente na aplicação. Por conseguinte mesmo que a aplicação seja reiniciada, os dados irão permanecer guardados e há a possibilidade de redirecionar o utilizador diretamente para a página inicial (caso este tenha o *login* efetuado).

Método	URL
POST	/user/login
Parâmetro	Descrição
email	Endereço de email do utilizador.
password	Password definida pelo utilizador.
Estado	Resposta
200	{"accepted": {idUser:"11c51c23-13d6-11ea"}}
401	{"error": "Email ou password incorretos."}
404	{"error": "O servidor está em baixo. Tente de novo."}
500	{"error": "Algo correu mal. Tente de novo."}

Tabela 3.2: Efetuar *Login*

O acesso aos dados do utilizador é realizado com um pedido GET que recebe como parâmetro o seu identificador, guardado no *AsyncStorage*. Se não ocorrerem erros é devolvida toda a informação do sistema relativa ao utilizador, como se constata na tabela 3.3.

Método	URL
GET	/user/profile
Parâmetro	Descrição
idUser	Identificador do Utilizador.
Estado	Resposta
200	<pre>{ "accepted": { "idUser": "11c51c23-13d6-11ea", "firstName": "John", "lastName": "Smith", "birthDate": "1995-12-21", "userPhoto": "iVBORw0KGgoAAAANSUhEU "password": "me7pÓ^", "documents": [{"type": "citizenCard", "documentNumber": "1234567", "expiryDate": "2022-03-01"}, {"type": "driverLicense", "documentNumber": "L-12345678", "expirationDate": "2022-03-01"}], "vehicles": [{"idVehicle": "93cbf6d5-1c68-11ea", "brand": "SEAT", "model": "ARONA", "enginePower": 2000, "engineType": "DIESEL", "amountSeats": 5, "photoVehicle": "iVBORw0KGgoAAAANSUhEUg", "color": "#000000", "registerDate": "2019-12-11", "vehicleType": "PASSENGERS", "beltforPets": true, "weight": 2000, "co2Emissions": 180, "vehiclePlate": "00-00-00", "nationalCategory": "B"}], "pets": [{"breedPet": "Saint Bernard", "electronicIdentifier": "0000012", "weight": 50, "size": 70, "photo": ""}]} </pre>
401	{"error": "Identificador do anúncio ou do utilizador incorreto."}
404	{"error": "O servidor está em baixo. Tente de novo."}
500	{"error": "Algo correu mal. Tente de novo."}

Tabela 3.3: Obter Perfil

A API de adaptação acede a um conjunto de métodos relativos aos anúncios, que são essenciais para a implementação da obtenção de informação de um anúncio, subscrição de um anúncio e rejeição de uma subscrição.

O acesso à informação sobre um anúncio é requisitado, por exemplo, quando um passageiro seleciona um dos carros para visualizar o destino. Como descrito na tabela 3.4, através do identificador do anúncio é possível obter toda a sua informação. Este anúncio é constituído pelo identificador do veículo e do condutor, pela hora de início, a origem e o destino e, por fim, pelos *waypoints*. Os *waypoints* contêm as posições dos passageiros a recolher.

Método	URL
GET	/announcement/
Parâmetro	Descrição
idAnnouncement	Identificador do Anúncio
Estado	Resposta
200	<pre>{ "accepted": { "announcement": { "idVehicle": "93cbf6d5-1c68-11ea", "idAnnouncement": "01665abe-1f4e-11ea", "idUser": "11c51c23-13d6-11ea", "beginningdateHour": "2020-04-12 17:22:19.75", "origin": { "lat": 38.721235, "long": -9.459913, "name": "Rua São João" }, "destiny": { "lat": 38.710464, "long": -9.446629, "name": "Rua da Torre" }, "waypoints": [{"lat": 38.717427, "long": -9.460535, "name": "Rua Santa Cruz"}] } } }</pre>
401	{"error": "Identificador de anúncio incorreto"}
500	{"error": "Algo correu mal. Tente de novo. "}

Tabela 3.4: Obter Anúncio

Como descrito nos casos de utilização citados na secção 3.1 deste capítulo, um passageiro pode subscrever a um anúncio de boleia e um condutor pode aceitar ou rejeitar esta subscrição. Relativamente à subscrição do lado do passageiro, recorre-se a um pedido POST que tem como parâmetros o identificador do passageiro e da viagem que este quer subscrever (tabela 3.5). Relativamente à decisão do condutor, é feito um pedido DELETE que necessita dos seguintes parâmetros: identificador do anúncio, identificador do passageiro e aceitar ou rejeitar a subscrição (tabela 3.6).

Método	URL
POST	/announcement/subscribe
Parâmetro	Descrição
idAnnouncement	Identificador do Anúncio
idUser	Identificador do Passageiro
Estado	Resposta
200	{"accepted": "Juntou-se ao anúncio."}
401	{"error": "Este anúncio está cheio."}
500	{"error": "Algo correu mal. Tente de novo."}

Tabela 3.5: Subscrever Anúncio

Método	URL
DELETE	/announcement/subscribe
Parâmetro	Descrição
idAnnouncement	Identificador do Anúncio.
idUser	Identificador do Passageiro.
consent	Aceitar ou rejeitar a subscrição.
Estado	Resposta
200	{"accepted": "A sua alteração foi registada"}
401	{"error": "Identificador do anúncio ou do utilizador incorreto."}
500	{"error": "Algo correu mal. Tente de novo."}

Tabela 3.6: Aceitar ou Rejeitar Subscrição

3.3 Simulação

Para ser possível demonstrar o uso desta aplicação no mundo real foi necessário desenvolver um mecanismo de simulação de viagens. Este mecanismo funciona com viagens pré-programadas associadas a um condutor fictício.

Deste modo, foram criados dois ficheiros JSON, estruturados de forma idêntica ao sistema boleias referenciado. Um dos ficheiros destina-se à informação dos utilizadores e o outro à informação dos anúncios. A estrutura de demonstração, ao ser desenvolvida desta forma, garante que o funcionamento da aplicação não será afetado ao trocar o acesso entre dados de simulação e do sistema.

Para obter a informação do trajeto das viagens foi necessário obter um ficheiro *Keyhole Markup Language* (KML) através do Google Maps, o qual contém informação relativa às coordenadas do trajeto.

As coordenadas obtidas são copiadas para um ficheiro txt e posteriormente formatadas através de um programa escrito em Java. Este programa coloca as coordenadas no formato "lat": 00000, "long": 00000 e gera um novo ficheiro, onde cada linha corresponde a uma posição do condutor fictício.

Cada linha é processada pelo servidor de quatro em quatro segundos e atri-

bui a coordenada ao condutor correspondente de modo a simular um carro em movimento.

Quando é efectuado um pedido de boleia a uma viagem simulada é feita uma alteração do caminho que está a ser realizado. Esta alteração é feita com recurso à API de direções do Google Maps que devolve o novo caminho após um pedido HTTP.

Capítulo 4

Demonstração

Para demonstrar o funcionamento do sistema implementado neste projeto e representar os casos de utilização numa situação real, foi concebido um cenário de exemplo para cada um dos utilizadores definidos (passageiro e condutor). Ambos os utilizadores estão registados no sistema de referência utilizado.

Cenários:

- Um utilizador passageiro de uma grande cidade, procura boleia para o seu local de trabalho.
- Um condutor que tinha uma viagem pré-programada, inicia a sua viagem na expectativa de que passageiros se juntem a esta, de forma a ocupar os lugares livres no carro.

Para efeitos de demonstração, todas as figuras relativas ao funcionamento da aplicação foram retiradas de simuladores de dispositivos Android. Os testes em dispositivos IOS foram realizados com recurso a dispositivos físicos.

4.1 Área Comum

Na figura 4.1 estão representados os ecrãs relativos à autenticação. Para o utilizador conseguir efetuar a autenticação através do login, é necessário ter uma conta registada no sistema de referência. Caso o login já tenha sido efetuado no dispositivo, o utilizador é direcionado para a página inicial de forma automática.

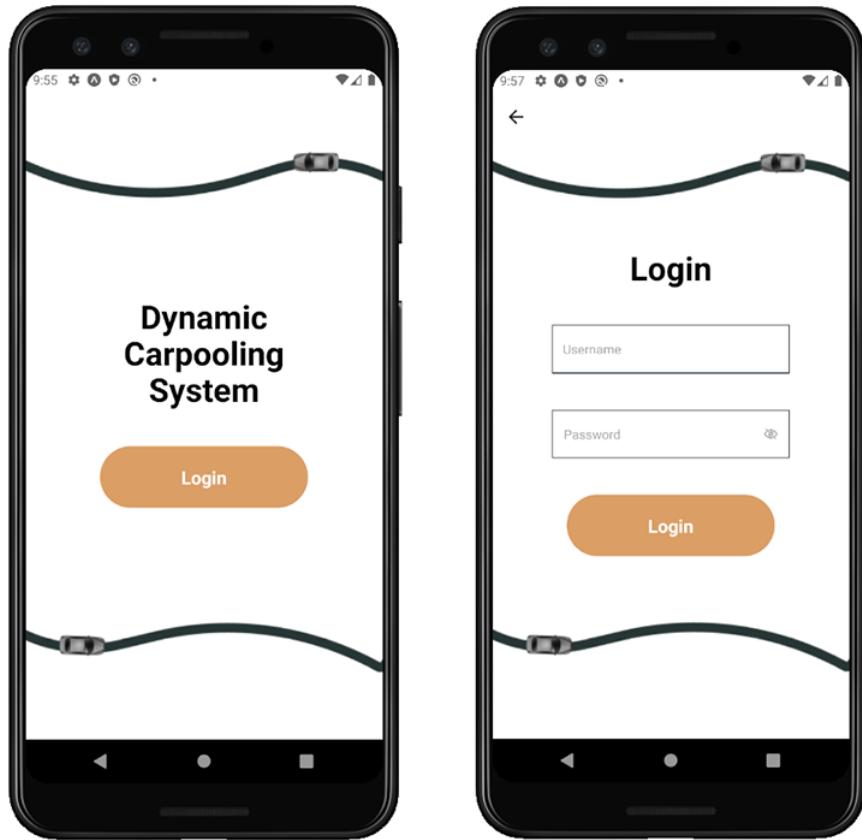


Figura 4.1: Autenticação

Após efetuar o login, consoante o perfil do utilizador, são apresentadas informações diferentes. Na página inicial, existem dois botões que podem ser apresentados (figura 4.2), o botão "Passageiro" e o botão "Condutor", cada um destes direciona para a página respetiva. Se for passageiro irá aparecer apenas o botão "Passageiro", se for condutor aparece o botão "Passageiro" e o botão "Condutor".

As diferenças retratadas na página inicial também podem ser observadas na página referente ao perfil de utilizador. A figura 4.3 demonstra como são apresentados os dados de um utilizador passageiro e a figura 4.4 os de um utilizador elegível a condutor onde são exibidas as informações relativas à carta de condução e veículo(s).

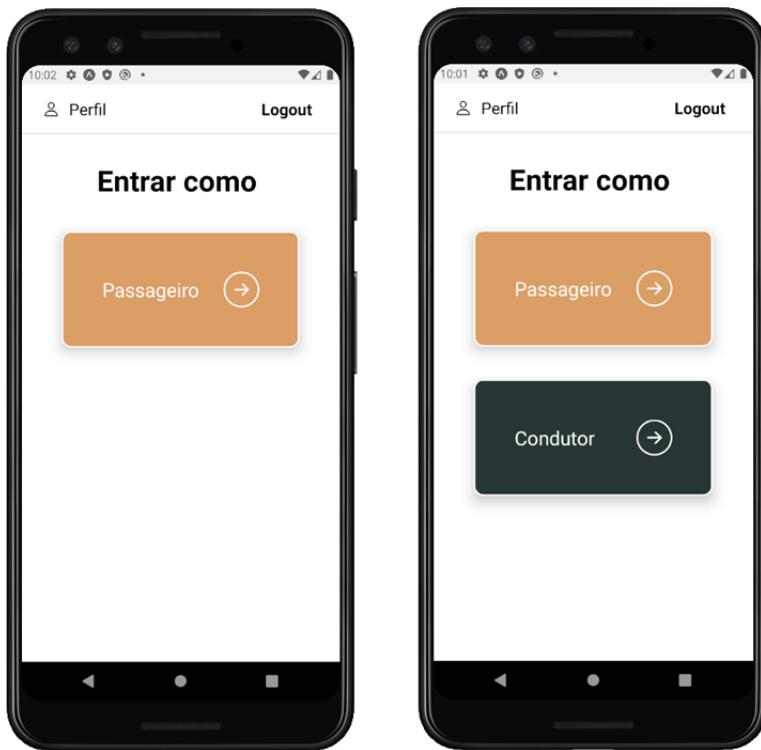


Figura 4.2: Página Inicial



Figura 4.3: Perfil - Apenas Passageiro

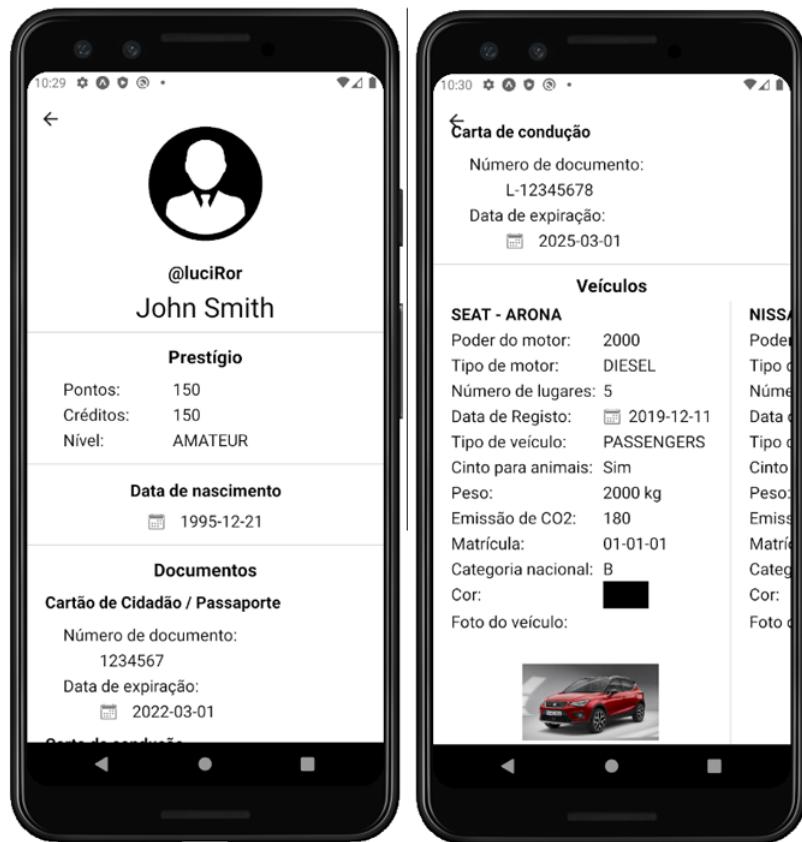


Figura 4.4: Perfil - Elegível a Condutor

4.2 Cenário Passageiro

Como descrito no primeiro cenário, um utilizador que pretenda ir para o seu local de trabalho poderá fazê-lo através da aplicação móvel desenvolvida.

Ao clicar no botão ”Passageiro”, o utilizador é reencaminhado para uma página com um mapa, onde são apresentadas todas as viagens que existem à sua volta, como se pode observar na figura 4.5.

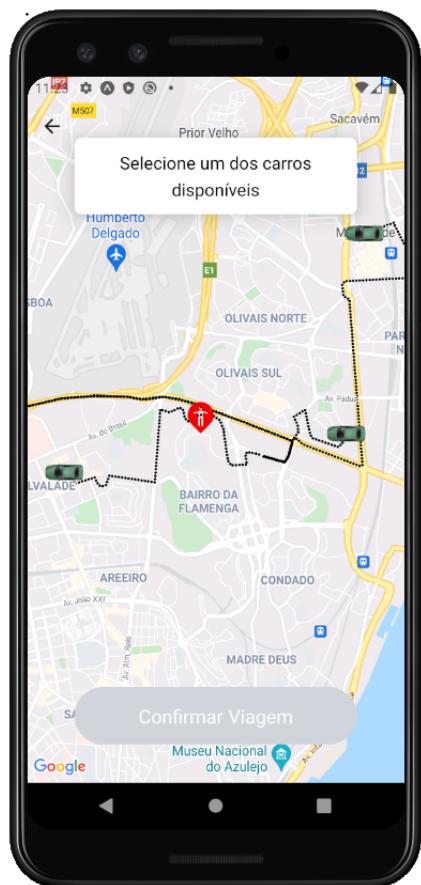


Figura 4.5: Ecrã de Passageiro

De seguida, ao selecionar um carro é mostrado o percurso que este está a realizar e o destino da viagem (figura 4.6). Se o passageiro vir que o destino do condutor corresponde, ou aproxima-se ao seu local de trabalho, pode efetuar uma subscrição. Ao efetuar esta subscrição é direcionado para um mapa com a sua localização atual e tem a possibilidade de arrastar o marcador para a localização exata onde pretende ser apanhado (figura 4.7). O passageiro confirma a posição e aparece um *pop-up* indicando que o condutor foi informado. Após a decisão do condutor, o *pop-up* é atualizado com a informação de sucesso ou de rejeição (figura 4.8).

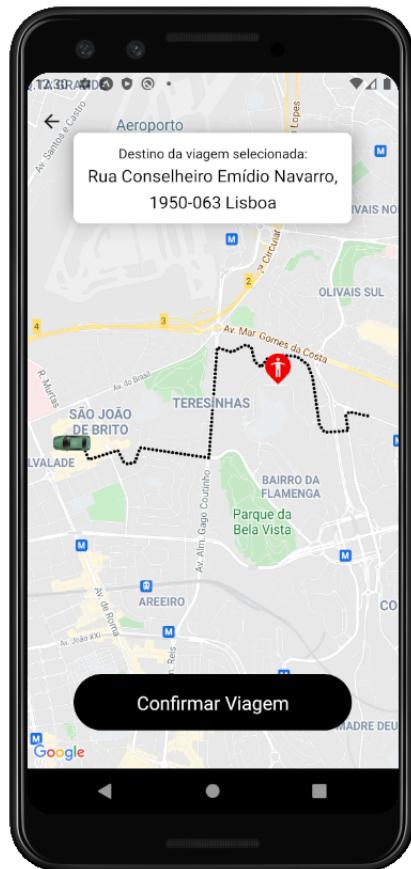


Figura 4.6: Viagem Selecionada

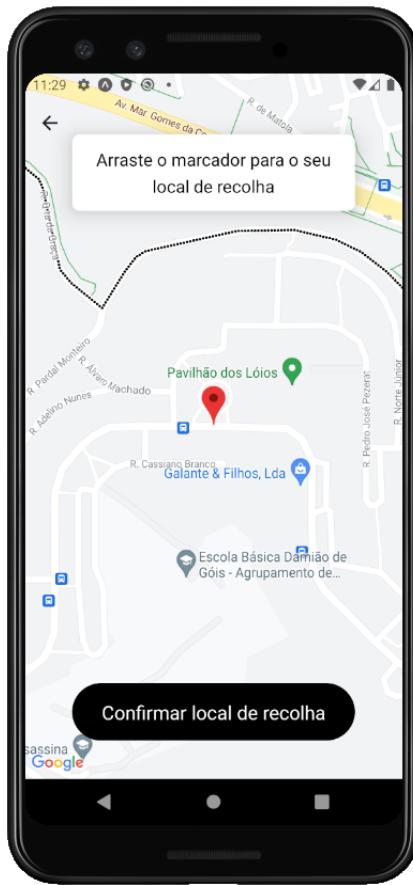


Figura 4.7: Recolha

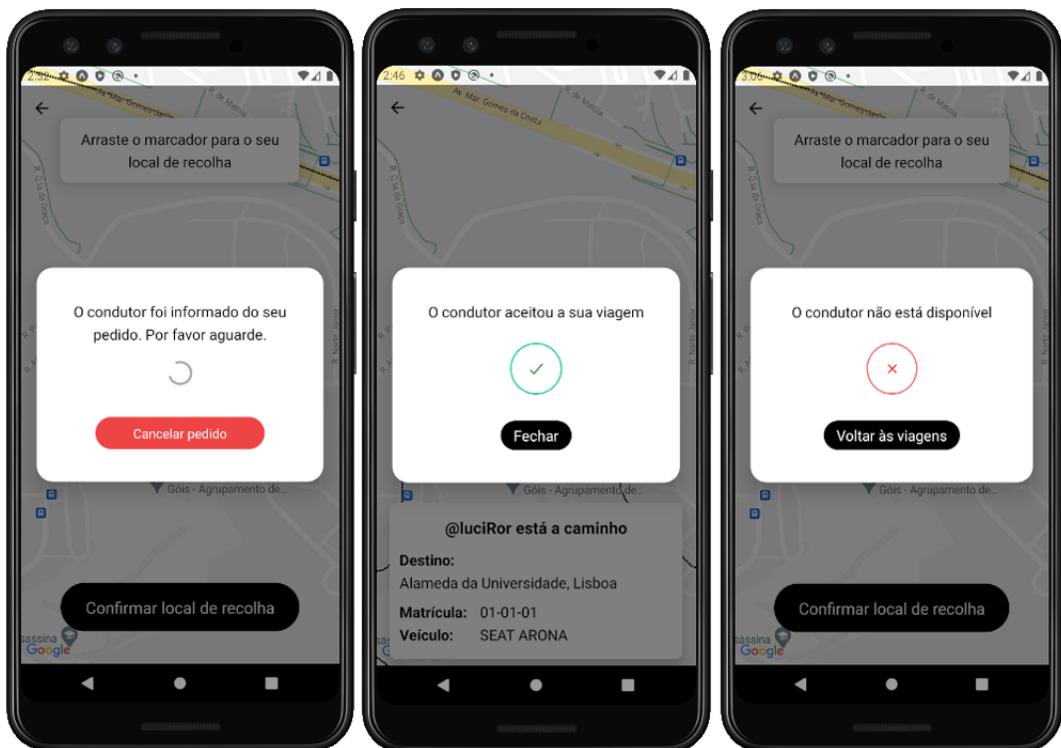


Figura 4.8: Resposta à Subscrição

No caso de o pedido ter sido aceite, o passageiro passa a visualizar a viagem com um ponto de paragem na sua localização e são dadas as informações do veículo que o vai recolher (figura 4.9).

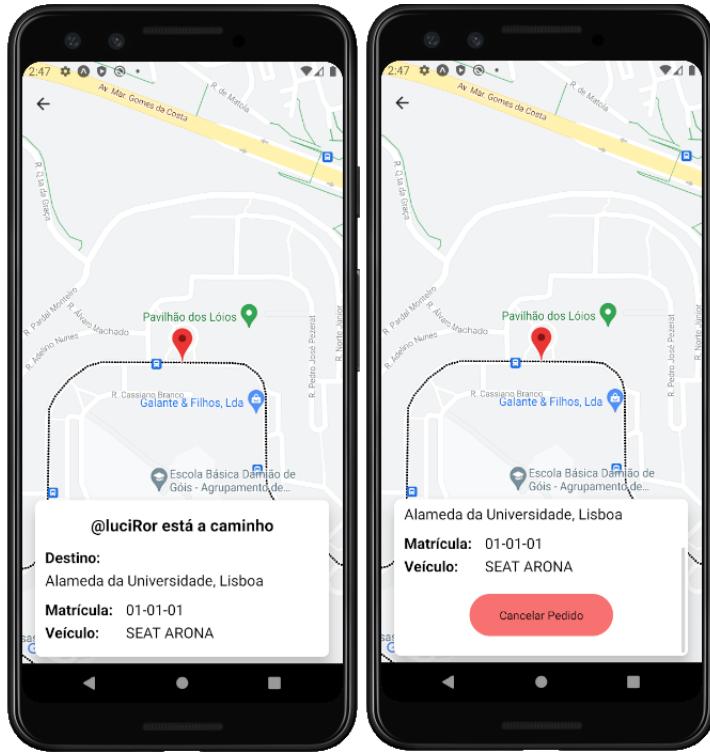


Figura 4.9: Viagem - Lado do Passageiro

4.3 Cenário Condutor

No segundo cenário, um utilizador que pretende iniciar a sua viagem pré-programada, entra no ecrã do condutor e caso a hora pré-programada corresponda à atual aparece um *pop-up* com as informações da viagem e poderá confirmar que a quer iniciar (figura 4.10).

Após iniciar a viagem, o mapa apresenta o carro a mover-se no percurso a realizar, de acordo com a sua posição. O condutor pode cancelar a sua viagem a qualquer momento e esta deixa de ficar disponível para os passageiros (figura 4.11).



Figura 4.10: Ecrã de condutor

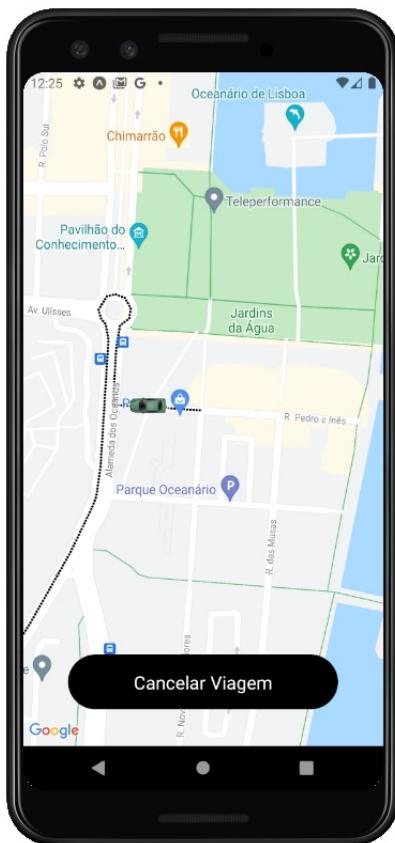


Figura 4.11: Viagem - Lado do Condutor

Se um passageiro efetuar um pedido de subscrição referente à viagem do condutor, aparece um *pop-up* no ecrã do condutor onde é apresentado o nome de utilizador do passageiro. Além disso, no mapa aparece um marcador com a localização do passageiro para verificar se este se encontra a uma distância razoável. Com base nestas informações, o condutor decide aceitar ou rejeitar a viagem (figura 4.12 lado esquerdo). Caso aceite a subscrição, o percurso da viagem é recalculado e adicionado o ponto de recolha do passageiro (figura 4.12 lado direito).

Se o condutor não tiver viagens marcadas na hora atual, ou cancelar a viagem em que se encontra, aparece um *pop-up* indicando que não existem viagens registadas àquela hora e é dada a opção de voltar à página inicial (figura 4.13).

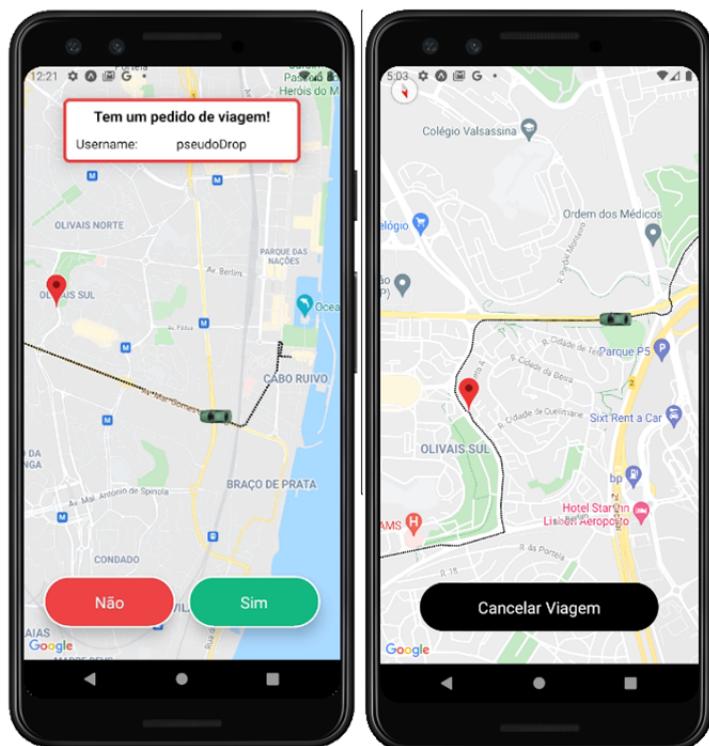


Figura 4.12: Pedido de Subscrição



Figura 4.13: Sem Viagem - Condutor

Capítulo 5

Conclusão

No presente capítulo é retratado o trabalho futuro e as considerações finais.

5.1 Trabalho Futuro

Seria interessante como desenvolvimento futuro, o passageiro poder adicionar mais informações no momento em que requisita a viagem, nomeadamente se viaja com um animal de estimação e a quantidade de bagagem que transporta consigo. E também poderia ser uma mais valia a implementação do sistema de classificação, suportado pelo sistema de referência, o que facilitaria a atribuição de uma classificação após o fim da viagem.

5.2 Considerações Finais

Apesar das limitações de tempo e de recursos disponibilizados os objetivos definidos neste projeto foram alcançados com sucesso. O domínio das diversas tecnologias utilizadas permitiu a criação de uma aplicação móvel para SO Android e IOS, desenvolvida em React Native. A camada de apresentação, no qual assenta o React Native, acede através de sockets a uma API em JavaScript, que foi hospedada num servidor Express do Node.js. A API desenvolvida encapsula um gestor de localizações responsável por gerir e enviar as posições dos condutores para os passageiros que as requisitem e uma API de adaptação que acede a outra API de serviços externa através de pedidos REST, o que possibilita a aquisição dos dados de um sistema de boleias existente.

O DCS implementado é fundamental para as comunidades de *Carpooling* porque facilita a requisição de boleias em qualquer lugar e a qualquer hora, sem planeamento prévio. Desta forma, a rede de utilizadores é expandida e as boleias que previamente poderiam não ser ocupadas na sua totalidade ou ocupadas de todo, passam a ter mais uma possibilidade de obtenção de passageiros. Com os veículos rentabilizados ao máximo, a quantidade de veículos privados nas estradas irá diminuir e consequentemente os níveis de poluição causados por este tipo de transporte. Que se traduz na preservação de recursos naturais e do meio ambiente.

No futuro, é esperado que este sistema de transporte se transforme num meio de deslocação sustentável que, simultaneamente, impulsiona a redução da poluição e aumenta a interação humana.

Bibliografia

- [1] SNS, ”OMS — Poluição atmosférica”, 02 maio 2018. [Online]. Disponível: <https://www.sns.gov.pt/noticias/2018/05/02/oms-poluicao-atmosferica/>
- [2] SNS, ”*The Effects of Climate Change*”, [Online]. Disponível: <https://climate.nasa.gov/effects/>
- [3] N. D. Chan e S. A. Shaheen, ”*Transport Reviews, Ridesharing in North America: Past, Present and Future*”, p. 20, 4 novembro 2011.
- [4] Rodrigo Diniz de Moura, ”Agregador de Sistemas de Boleias”, outubro 2020, [Online]. Disponível: <https://repositorio.ipl.pt/handle/10400.21/12479>
- [5] WAZE, [Online]. Disponível: <https://www.waze.com/pt-BR/carpool>
- [6] BLABLACAR, [Online]. Disponível: <https://www.blablacar.pt>
- [7] SCOOP, [Online]. Disponível: <https://takescoop.zendesk.com/hc/en-us>
- [8] Sergey Korolev, *Mobile App Development Approaches Explained*, 19 junho 2019, [Online]. Disponível: <https://railsware.com/blog/native-vs-hybrid-vs-cross-platform/>
- [9] REACT NATIVE, [Online]. Disponível: <https://reactnative.dev>
- [10] FLUTTER, [Online]. Disponível: <https://flutter.dev>
- [11] XAMARIN, [Online]. Disponível: <https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin>
- [12] EGO, *Flutter VS Xamarin VS React Native: What's Best in 2021?*, 26 janeiro 2021, [Online]. Disponível: <https://www.ego-cms.com/post/flutter-vs-xamarin-vs-react-native-whats-best-in-2021>

- [13] ORACLE, “*REST API Methods*”, [Online]. Disponível: https://docs.oracle.com/en/cloud/saas/enterprise-performance-management-common/prest/rest_api_methods.html
- [14] MOZILLA, “*Express Web Framework (Node.js/JavaScript)*”, [Online]. Disponível: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs
- [15] SOCKET.IO, “*What Socket.IO is*”, [Online]. Disponível: <https://socket.io/docs/v4/>