



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, com o propósito de facilitar o desacoplamento e a portabilidade face a outros SB.

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 arises. This system 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 with the purpose of facilitating decoupling and portability compared to other carpooling systems

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	4
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

2.2	Tecnologias Multiplataforma	9
2.2.1	React Native	9
2.2.2	Flutter	9
2.2.3	Xamarin	9
2.3	Tecnologias <i>Backend</i>	10
2.3.1	Node.js	10
2.3.2	Pyhton	11
2.4	Comunicação	11
2.4.1	WebSockets	11
2.4.2	Socket.IO	11
3	Projeto	13
3.1	Escolha das Tecnologias	13
3.1.1	Tipo de Aplicação	13
3.1.2	Tecnologia Multiplataforma	13
3.1.3	Tecnologia <i>Backend</i>	14
3.1.4	Comunicação	14
3.2	Casos de Utilização	14
3.3	Arquitetura	15
3.3.1	Aplicação	16
3.3.2	Serviços	17
3.3.3	Recursos	22
4	Demonstração	29
4.1	Simulação	29
4.2	Área Comum	30
4.3	Cenário Passageiro	33
4.4	Cenário Condutor	36
5	Conclusão	39
5.1	Trabalho Futuro	39
5.2	Considerações Finais	39
	Bibliografia	41

Lista de Tabelas

2.1	Tipos de Aplicação - Vantagens e Desvantagens	8
2.2	<i>Frameworks</i> - Vantagens e Desvantagens	10
3.1	Iniciar Viagem	17
3.2	Terminar Viagem	18
3.3	Atualizar Localização do Condutor	18
3.4	Condutores em Viagem	19
3.5	Viagem a Decorrer	19
3.6	Emissão da Subscrição	20
3.7	Emissão da Resposta à Subscrição	20
3.8	Pedido de Resposta a Subscrição	21
3.9	Informação de uma boleia a decorrer	21
3.10	URLs de Acesso à Máquina Virtual	22
3.11	Efetuar <i>Login</i>	23
3.12	Obter Perfil	24
3.13	Obter Anúncio	25
3.14	Subscrever Anúncio	26
3.15	Aceitar ou Rejeitar Subscrição	27

Lista de Figuras

1.1	Ecrãs Waze Carpool	4
1.2	Ecrãs BlaBlaCar	4
1.3	Ecrãs Scoop	5
3.1	Casos de Utilização	15
3.2	Arquitetura DCS	16
4.1	Autenticação	31
4.2	Página Inicial	31
4.3	Perfil - Apenas Passageiro	32
4.4	Perfil - Elegível a Condutor	32
4.5	Ecrã de Passageiro	33
4.6	Viagem Selecionada	34
4.7	Recolha	34
4.8	Resposta à Subscrição	35
4.9	Viagem - Ecrã do Passageiro	35
4.10	Ecrã de condutor	36
4.11	Viagem - Ecrã do Condutor	37
4.12	Pedido de Subscrição	38

Listas de Acrônimos

API *Application Programming Interface*

CSS *Cascading Style Sheets*

CO2 Dióxido de Carbono

DCS *Dynamic Carpooling System*

GPS *Global Positioning System*

HTTP *Hypertext Transfer Protocol*

JSON *JavaScript Object Notation*

KML *Keyhole Markup Language*

OMS Organização Mundial de Saúde

REST *Representational State Transfer*

SB *Sistema de Boleias*

SO Sistemas Operativos

URL *Uniform Resource Locator*

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 automóveis nos dias de hoje é a origem de muita da poluição atmosférica existente. Este tipo de poluição traduz-se na presença de substâncias nocivas na atmosfera e representa um problema grave para a saúde pública, dado que pode desencadear problemas respiratórios e cardiovasculares. 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 nocivos relacionados com o meio ambiente tal como o aquecimento global. O aquecimento global define-se como o processo de aumento da temperatura média dos oceanos e da atmosfera, tendo como consequências, por exemplo, 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 particular. As viagens estão organizadas, geralmente, em comunidades com interesses comuns e os seus membros podem eventualmente trabalhar na mesma empresa, morar na mesma localidade ou ir ao mesmo evento (cultural, desportivo). 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. Relativamente às vantagens, este sistema incentiva a sustentabilidade ambiental, reduz as emissões de CO₂ emitidas pelos veículos automóveis, é 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.

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 automóveis devido à sua diminuição nas estradas. Não sendo a solução para diminuir toda a poluição, é um passo em frente na sua redução e na consciencialização da população, possui vantagens económicas, sociais e de sustentabilidade ambiental.

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 permita subscrever boleias em curso usando um dispositivo móvel.

Através da solução informática implementada espera-se divulgar e promover este sistema de *Carpooling* para diminuir a poluição que os transportes particulares causam.

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* [4], que pode ser executada, por exemplo, nos Sistemas Operativos (SO) Android [5] ou IOS [6].

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 Boleias (SB) de referência utilizado na dissertação intitulada Agregador de Sistemas de Boleias [7].

1.5 Plataformas Relacionadas

À semelhança de outros sistemas conhecidos, foram desenvolvidas, ao longo dos anos, plataformas que suportam o *Carpooling*. As plataformas privadas descritas nesta secção, atualmente, não suportam a requisição de boleias de uma viagem que está a decorrer.

1.5.1 Plataformas Privadas

Waze Carpool

O Waze [8] é uma aplicação para dispositivos móveis, baseada na navegação por *Global Positioning System* (GPS) e suportada pela Google [9]. 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: Ecrãs Waze Carpool

BlaBlaCar

BlaBlaCar [10] 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 une condutores e passageiros e pode ser acedida através da versão *Web* ou aplicação móvel.

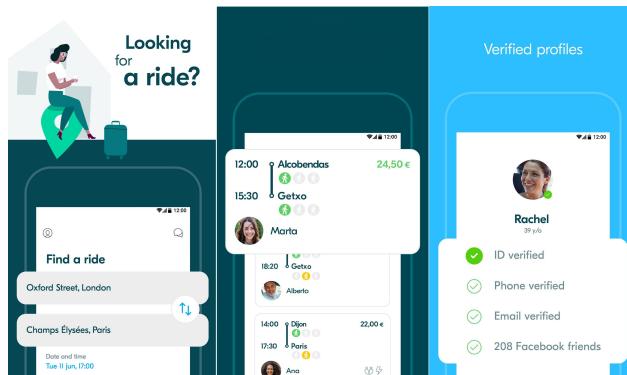


Figura 1.2: Ecrãs BlaBlaCar

Scoop

Fundado em 2015, o Scoop [11] une 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 [12], FedEx [13] e Samsung [14], reafirmando a sua segurança. À semelhança do Waze Carpool ainda não está disponível em Portugal.

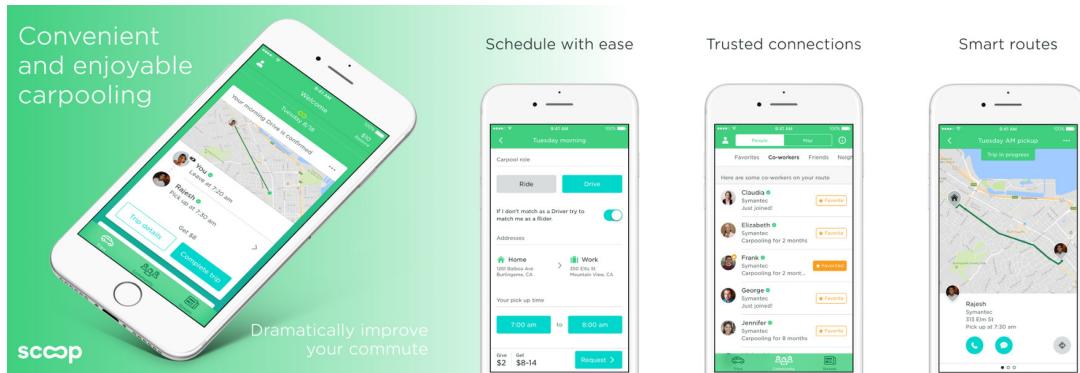


Figura 1.3: Ecrãs Scoop

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” [7].

1.6 Organização do Documento

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

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

O terceiro capítulo, apresenta o projeto, onde são evidenciadas as tecnologias escolhidas na implementação, descritos os casos de utilização, e é definida a arquitetura implementada.

O quarto capítulo constitui a demonstração, é apresentado um conjunto de exemplos que demonstram as funcionalidades descritas nos casos de utilização.

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

Capítulo 2

Tecnologias

Face à natureza deste projeto, é essencial que um SB esteja acessível a partir de qualquer lugar. Sendo assim, foram avaliadas tecnologias que suportam o desenvolvimento para dispositivos móveis, dispositivos estes que operam na sua maioria com SO Android ou IOS.

2.1 Tipos de Aplicação

No contexto deste projeto foram considerados três tipos de aplicações móveis [15] 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 [16] ou Swift [17] enquanto para Android são desenvolvidas em Java [18] ou Kotlin [19].

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

Uma aplicação híbrida é desenvolvida com a mesma tecnologia utilizada no desenvolvimento de aplicações para a *web* e é executada na infraestrutura nativa do dispositivo. Como uma versão nativa, uma aplicação híbrida pode incorporar recursos do sistema operativo (Android ou IOS) e usar tecnologias *web* compatíveis (HTML5 [20], *Cascading Style Sheets* (CSS) [21] e JavaScript [22]).

	Vantagens	Desvantagens
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
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 • Menos custos de desenvolvimento 	<ul style="list-style-type: none"> • Dificuldade de integração • Pior performance • Acesso mais restrito a funções do sistema (menos limitado que no híbrido)
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

Tabela 2.1: Tipos de Aplicação - Vantagens e Desvantagens

2.2 Tecnologias Multiplataforma

No âmbito do desenvolvimento multiplataforma foram consideradas três das *frameworks* mais conhecidas e utilizadas atualmente, sendo estas:

2.2.1 React Native

O React Native [4] é um *framework* Javascript desenvolvido pelo Facebook [23] e é utilizado para desenvolver aplicações para os sistemas Android e IOS. A comunicação entre o React Native e os elementos nativos dos sistemas Android e IOS é realizada através de uma ”ponte” JavaScript, deste modo, o React consegue renderizar os seus elementos nativamente.

2.2.2 Flutter

O Flutter [24] é um *framework* desenvolvido pelo Google que utiliza a linguagem de programação Dart [25] 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 [26] é um *framework* multiplataforma desenvolvido pela Microsoft [27]. Tem o propósito de desenvolver aplicações para os SO Android, IOS e Windows [28] através da linguagem C# [29] e com .NET [30]. O Xamarin é uma camada de abstração que gera a comunicação entre código partilhado e código de plataforma subjacente.

	Vantagens	Desvantagens
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
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
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 Tecnologias *Backend*

Relativamente às tecnologias *backend* foram considerados o Node.js [31] e o Python [32].

2.3.1 Node.js

O Node.js é uma das tecnologias *backend* mais utilizadas e é uma forma de desenvolver ambientes leves e flexíveis. Está escrito em C, C++ [33], e JavaScript, e é executado no motor V8 [34] JavaScript *runtime*. Tem como vantagem uma arquitetura de entrada e saída não bloqueante, indicado para uma aplicação em tempo real. Além disto, disponibiliza várias ferramentas para o desenvolvimento de aplicações móveis e *web*.

2.3.2 Pyhton

O Python é uma linguagem de programação aberta, orientada a objetos, de alto nível e dinâmica. Devido à sua versatilidade pode ser utilizado tanto como tecnologia *backend*, como *frontend* e é muito aplicado no desenvolvimento de programas que requerem a aprendizagem automática ou análise de dados.

2.4 Comunicação

Foram consideradas duas tecnologias que possibilitam a comunicação entre frontend e backend, designadamente, WebSockets e Socket.IO [35].

2.4.1 WebSockets

O WebSocket é um protocolo que permite uma comunicação bidirecional entre cliente e servidor através de uma conexão TCP. Neste tipo de comunicação tanto o cliente como o servidor podem comunicar e enviar informação simultaneamente, proporcionando uma comunicação em tempo real.

2.4.2 Socket.IO

A biblioteca Javascript Socket.IO 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*. O Socket.IO tem ainda a capacidade de restabelecer a conexão automaticamente caso esta falhe.

Capítulo 3

Projeto

Este capítulo engloba os diferentes componentes do projeto, nomeadamente, a escolha das tecnologias, os casos de utilização e a arquitetura da solução informática delineada.

3.1 Escolha das Tecnologias

Aquando da implementação do presente projeto foram consideradas e analisadas as diferentes tecnologias apresentadas no capítulo 2.

3.1.1 Tipo de Aplicação

Uma aplicação nativa não foi considerada como opção viável devido aos elevados custos de produção baseados na necessidade de haverem duas equipas de desenvolvimento para os diferentes SO Android e IOS.

Após analisadas as vantagens e desvantagens (tabela 2.1, capítulo 2) das aplicações multiplataforma e híbridas, a escolha do tipo de aplicação a desenvolver assentou numa aplicação multiplataforma. Este tipo de aplicação é menos limitado no que toca ao acesso de funções do sistema que irão ser necessários, como por exemplo, o GPS dos utilizadores.

3.1.2 Tecnologia Multiplataforma

Após analisadas as três *frameworks* multiplataforma, através das vantagens e desvantagens descritas na tabela 2.2 (capítulo 2), 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* completa em termos de documentação, sendo complementada por um vasto leque de bibliotecas externas.

3.1.3 Tecnologia *Backend*

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 é simplificado uma vez que a heterogeneidade das tecnologias desenvolvidas diminui. Por este motivo e tendo em conta as vantagens relacionadas com o tempo de execução optou-se por utilizar o Node.js na implementação do servidor, com recurso à *framework* Express [36].

3.1.4 Comunicação

No que diz respeito à comunicação foi escolhida a biblioteca Socket.IO.

3.2 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 da carta de condução e de um automóvel registado 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 autenticar-se e, caso seja feito com sucesso, é autorizado o acesso. De seguida, os utilizadores podem consultar seu perfil que varia consoante se é passageiro ou condutor. Se for condutor 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 durante a viagem.

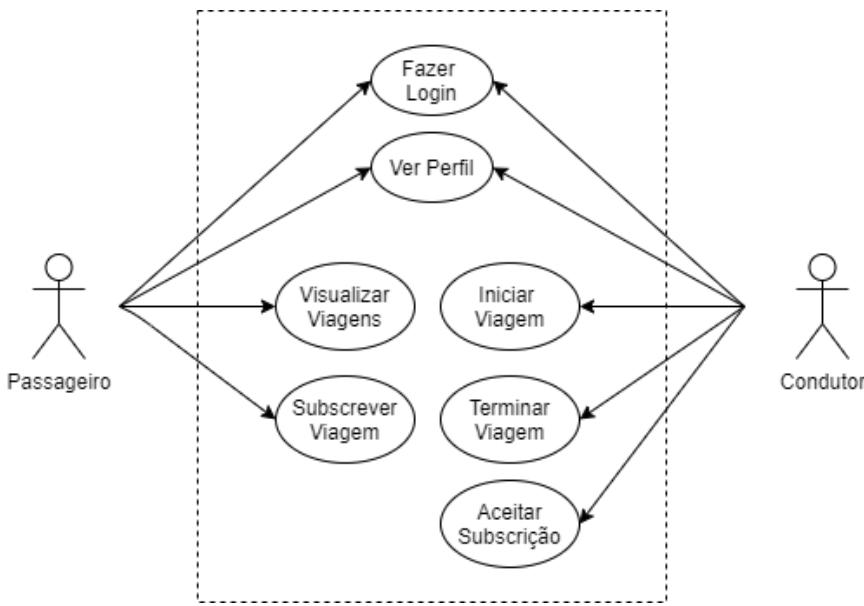


Figura 3.1: Casos de Utilização

3.3 Arquitetura

Um dos passos mais importantes, senão o mais importante, no desenvolvimento de um projeto é a conceção de uma arquitetura. 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, nomeadamente, camada de aplicação, camada de serviços e camada de recursos.

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

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

Por fim, existe a camada de recursos que é composta pela API exportada pelo sistema de referência utilizado na dissertação Agregador de Sistema de Boleias [7]. Esta camada é responsável por fornecer o acesso aos dados armazenados no SB.

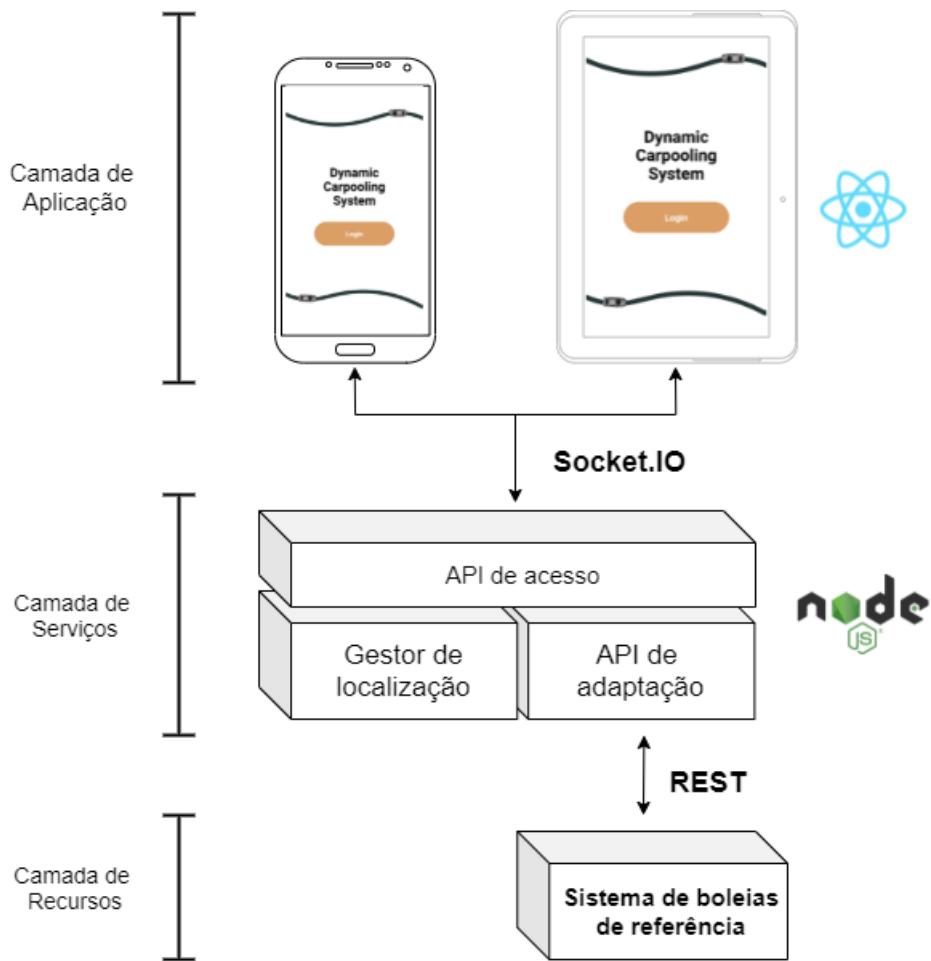


Figura 3.2: Arquitetura DCS

3.3.1 Aplicação

A camada de aplicação 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.

3.3.2 Serviços

A camada de serviços 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 a obtenção dos dados de dados da API de serviços do sistema de referência, demonstrado na figura 3.1 e o gestor de localização em tempo real.

Com recurso ao Socket.IO, foi estabelecido um conjunto de serviços que facilitam a comunicação entre a camada de serviços e a camada de aplicação. Estes serviços são implementados através de um mecanismo do Socket.IO que tem como base a classe *EventEmitter* do Node.js [37]. Este mecanismo recebe como primeiro parâmetro o nome do evento, seguido pelo seu *listener* (função responsável pelo processamento da informação recebida).

Quando um condutor pretende dar início a uma viagem, é emitido um evento ”*loadDriverInfo*” que, ao enviar os parâmetros referidos na tabela 3.1, comunica ao server qual o anúncio que está a começar, o socket por onde devem ser efectuados os pedidos de boleia e a localização atual do condutor.

Nome do evento:	loadDriverInfo
Parâmetro	Descrição
idAnnouncement	Identificador do Anúncio.
socketID	Identificador do socket
MyLocation	Localização atual do condutor

Tabela 3.1: Iniciar Viagem

Quando uma viagem chega ao seu término ou quando o condutor a cancela, o evento ”*endTrip*” é emitido (tabela 3.2. O serviço ”*endTrip*” tem como função eliminar a informação relativa a uma viagem que é visível aos passageiros. Se a viagem for cancelada, os utilizadores não poderão realizar uma subscrição da mesma.

Nome do evento:	endTrip
Parâmetro	Descrição
idAnnouncement	Identificador do Anúncio.

Tabela 3.2: Terminar Viagem

Na tabela 3.3 está descrita a estrutura do serviço denominado ”*update location*”. O serviço referido é emitido por um condutor que esteja a disponibilizar uma viagem e atualiza os dados relativos à sua localização.

Nome do evento:	update location
Parâmetro	Descrição
idAnnouncement	Identificador do Anúncio.
location(latlng)	Localização atual definida por um conjunto latitude, longitude.

Tabela 3.3: Atualizar Localização do Condutor

O serviço ”*get drivers*”, representado na tabela 3.4, possibilita que um passageiro ao entrar no ecrã relativo às viagens, visualize no mapa as viagens que estão a decorrer.

Nome do evento:	get drivers
Resposta	Descrição
drivers_connected	Array com informação dos anúncios das viagens a decorrer.
realTimeInfo	Array composto por objetos JSON que encapsulam a informação da localização de cada condutor, o seu socket e o identificador do anúncio da viagem que está a decorrer.

Tabela 3.4: Condutores em Viagem

O serviço ”*get driver by ID*” devolve a informação relativa a uma viagem a decorrer (tabela 3.5). Esta informação possibilita a visualização da viagem por um passageiro que a escolha.

Nome do evento:	get driver by ID
Parâmetro	Descrição
idAnnouncement	Identificador do Anúncio.
Resposta	Descrição
realTimeInfo	Objeto JSON com identificador do anúncio, localização atual do condutor e o seu socket.

Tabela 3.5: Viagem a Decorrer

O serviço ”*pickup*”, descrito na tabela 3.6, é emitido por um passageiro ao efetuar uma subscrição e tem como objetivo alertar o condutor que está a realizar a viagem.

Nome do evento:	pickup
Parâmetro	Descrição
idAnnouncement	Identificador do Anúncio.
PickupLocation	Localização de recolha do passageiro
SocketID	Identificador do socket
UserID	Identificador do utilizador
Resposta	Descrição
LastLocation	Última posição do condutor

Tabela 3.6: Emissão da Subscrição

Após receber um pedido de subscrição, o condutor emite o evento ”*tripResponse*”. Neste evento indica se aceita ou rejeita a subscrição do passageiro, como se pode observar na tabela 3.7.

Nome do evento:	tripResponse
Parâmetro	Descrição
idAnnouncement	Identificador do Anúncio.
Response	Resposta do condutor ao pedido

Tabela 3.7: Emissão da Resposta à Subscrição

Para verificar se o condutor já respondeu ao pedido de subscrição, o passageiro emite o evento ”*getTripReponse*” (tabela 3.8).

Nome do evento:	gettripResponse
Parâmetro	Descrição
socketID	Identificador do socket do anúncio
Resposta	Descrição
Answered	Variável que indica se o condutor respondeu ao pedido
Response	Resposta do condutor ao pedido

Tabela 3.8: Pedido de Resposta a Subscrição

Após a aprovação da subscrição, o passageiro acede à informação acerca da viagem através do evento ”*getTripInfo*”. Como se pode observar na tabela 3.9, o passageiro irá ter acesso à informação do veículo, de modo a facilitar o seu encontro.

Nome do evento:	getTripInfo
Parâmetro	Descrição
idAnnouncement	Identificador do Anúncio.
Resposta	
Username	Nome de utilizador do condutor
Plate	Matrícula do veículo
Vehicle	Modelo do veículo

Tabela 3.9: Informação de uma boleia a decorrer

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

3.3.3 Recursos

Como é possível observar na figura 3.2, foi feito um acesso a um SB de referência existente através de uma camada de adaptação. Ao reaproveitar um SB existente é possível utilizar na camada de recursos 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) [38], 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 3.10, na secção referente à interface de serviços, estão descritos os *Uniform Resource Locator* (URL) 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 3.10: URLs de Acesso à Máquina Virtual

Ao iniciar a aplicação móvel o utilizador é reencaminhado para o *login*. Após inserir as suas credenciais é utilizado o método POST para realizar a autenticação das mesmas. Na tabela 3.11 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* [39] 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 3.11: 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 3.12.

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"}, {"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 3.12: 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 3.13, através do identificador do anúncio é possível obter toda a sua informação. O 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 3.13: 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 3.14). 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 3.15).

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 3.14: 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 incorrecto."}
500	{"error": "Algo correu mal. Tente de novo."}

Tabela 3.15: Aceitar ou Rejeitar Subscrição

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 emuladores de dispositivos Android. Os testes em dispositivos IOS foram realizados com recurso a dispositivos físicos.

4.1 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 [40], 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 atribui 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.

4.2 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.

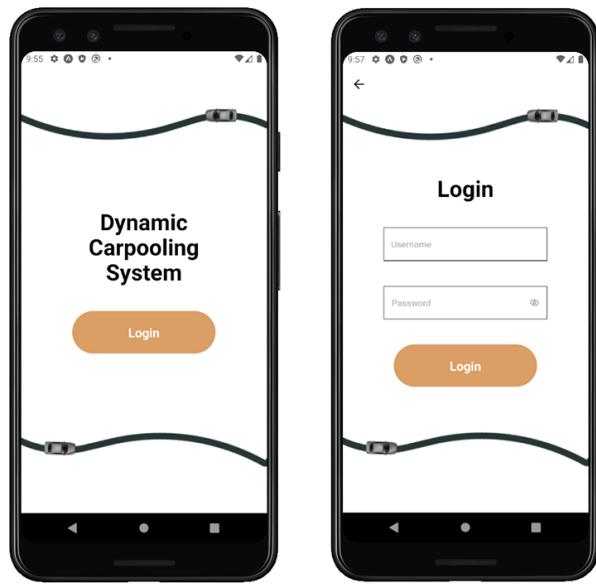


Figura 4.1: Autenticação

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”.

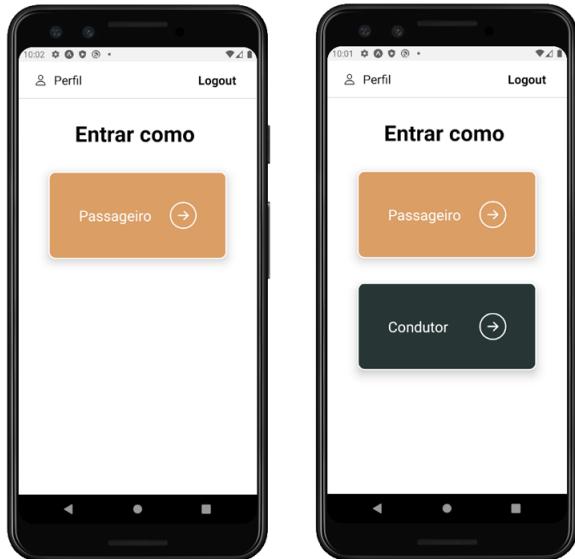


Figura 4.2: Página Inicial

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 4.3: Perfil - Apenas Passageiro

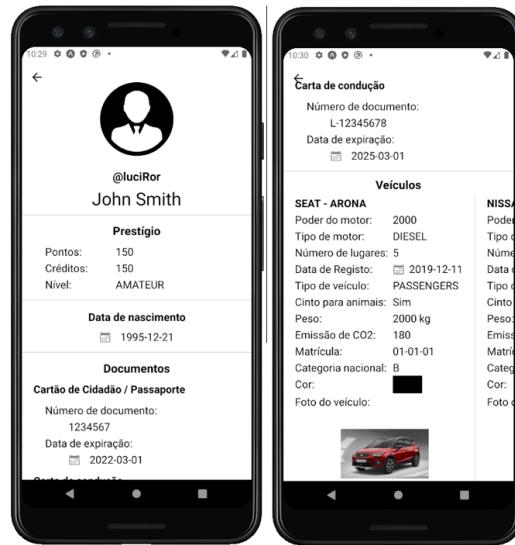


Figura 4.4: Perfil - Elegível a Condutor

4.3 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.

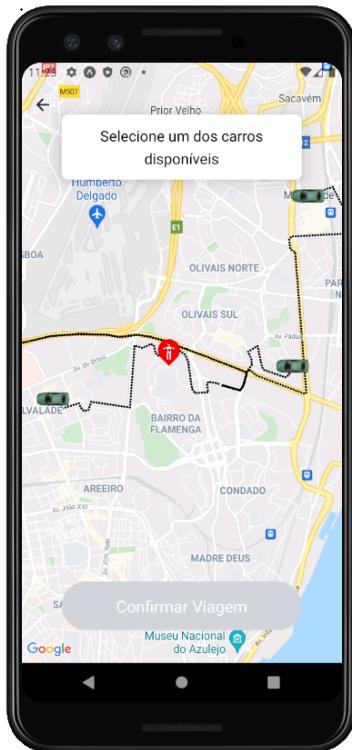


Figura 4.5: Ecrã de Passageiro

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).

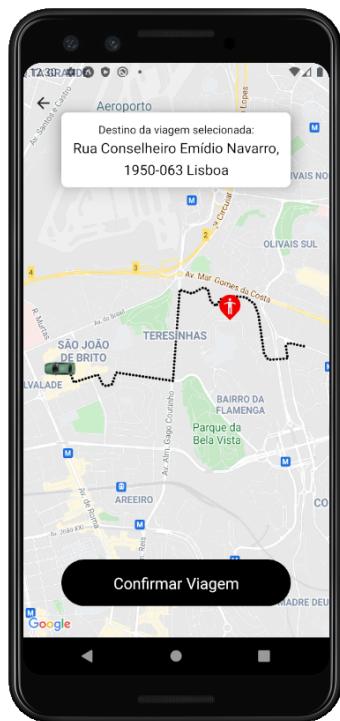


Figura 4.6: Viagem Selecionada



Figura 4.7: Recolha

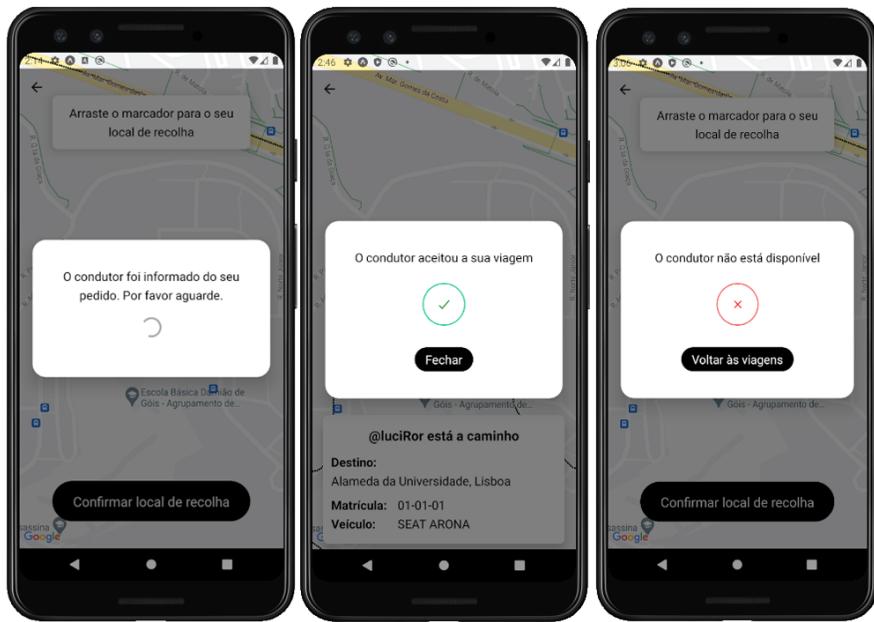


Figura 4.8: Resposta à Subscrição

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).

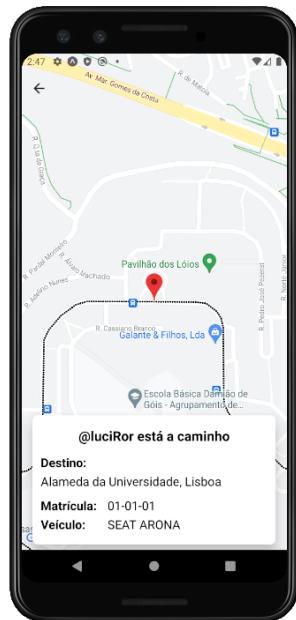


Figura 4.9: Viagem - Ecrã do Passageiro

4.4 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 lado esquerdo).

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 registradas àquela hora e é dada a opção de voltar à página inicial (figura 4.10 lado direito).

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).

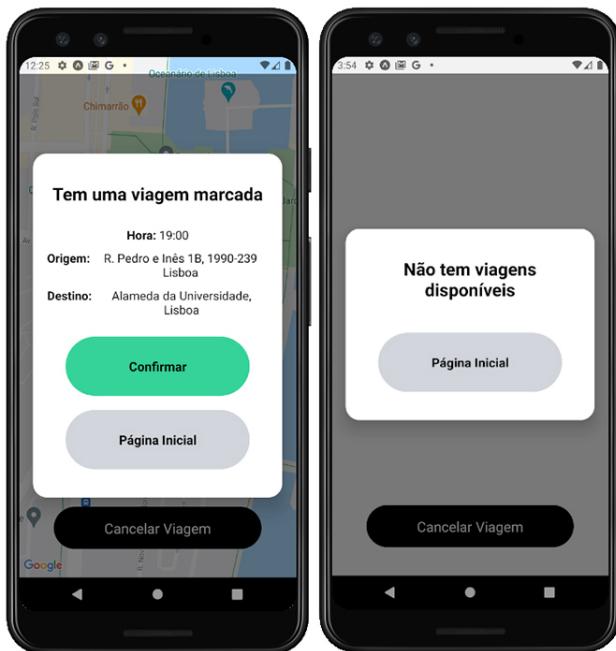


Figura 4.10: Ecrã de condutor

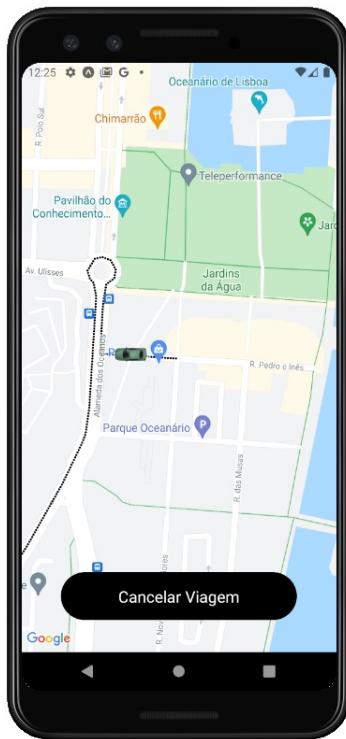


Figura 4.11: Viagem - Ecrã do Condutor

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).

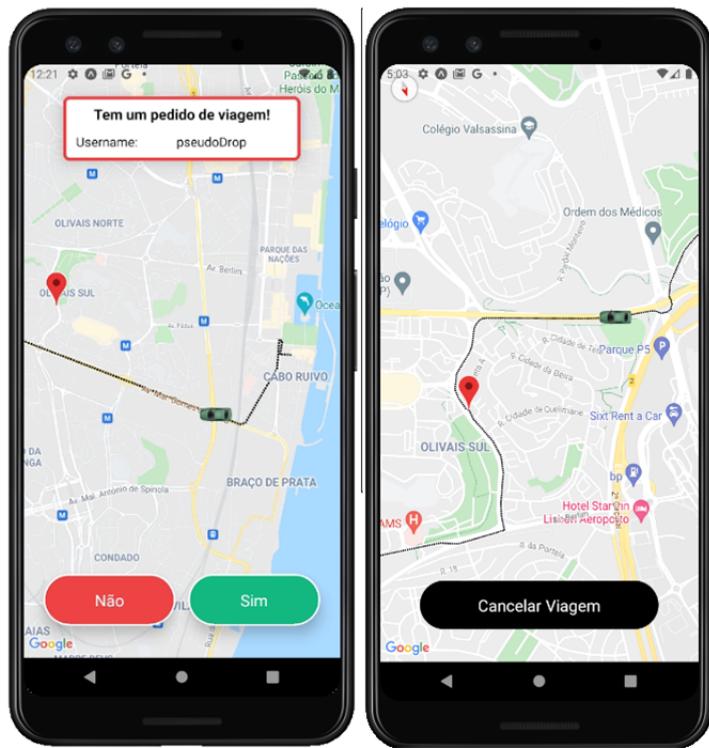


Figura 4.12: Pedido de Subscrição

Capítulo 5

Conclusão

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

5.1 Trabalho Futuro

Seria interessante como desenvolvimento futuro 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 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 aplicação, 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.

Através do DCS desenvolvido, um utilizador passageiro ou um utilizador elegível a condutor registado no SB de referência pode efetuar a autenticação na aplicação e ver o seu perfil. Um passageiro pode ver as viagens em curso nas redondezas e após selecionar a que lhe parece mais relevante, pode subscrevê-la. Um condutor

com uma viagem marcada previamente, poderá abrir o ecrã de condutor à hora marcada e iniciar viagem. Ao longo da sua viagem pode ver o seu caminho a atualizar com a sua posição e caso um passageiro efetue uma subscrição, o condutor poderá negar ou aceitar a mesma. Caso aceite, a viagem é redirecionada para passar a ter como ponto de passagem a localização do passageiro. No ecrã do passageiro, caso o condutor aceite a sua subscrição, o passageiro é informado de que o condutor está a caminho e são fornecidas as informações sobre o veículo.

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] NASA, "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] React Native, [Online]. Disponível: <https://reactnative.dev>
- [5] Android, [Online]. Disponível: <https://www.android.com>
- [6] IOS, [Online]. Disponível: <https://www.apple.com/pt/ios/ios-15/>
- [7] Rodrigo Diniz de Moura, "Agregador de Sistemas de Boleias", outubro 2020, [Online]. Disponível: <https://repositorio.ipl.pt/handle/10400.21/12479>
- [8] Waze, [Online]. Disponível: <https://www.waze.com/pt-BR/carpool>
- [9] GOOGLE, [Online]. Disponível: <https://about.google>
- [10] BlaBlaCar, [Online]. Disponível: <https://www.blablacar.pt>
- [11] Scoop, [Online]. Disponível: <https://takescoop.zendesk.com/hc/en-us>
- [12] LinkedIn, [Online]. Disponível: <https://www.linkedin.com>
- [13] FedEx, [Online]. Disponível: <https://www.fedex.com/pt-pt/home.html>
- [14] Samsung, [Online]. Disponível: <https://www.samsung.com/pt/>

- [15] Sergey Korolev, *Mobile App Development Approaches Explained*, 19 junho 2019, [Online]. Disponível: <https://railsware.com/blog/native-vs-hybrid-vs-cross-platform/>
- [16] Objective-C, [Online]. Disponível: <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>
- [17] Swift, [Online]. Disponível: <https://developer.apple.com/swift>
- [18] Java, [Online]. Disponível: https://www.java.com/pt-BR/about/whatis_java.jsp
- [19] Kotlin, [Online]. Disponível: <https://developer.android.com/kotlin>
- [20] HTML5, [Online]. Disponível: <https://developer.mozilla.org/en-US/docs/Glossary/HTML5>
- [21] CSS, [Online]. Disponível: <https://developer.mozilla.org/pt-BR/docs/Web/CSS>
- [22] JavaScript, [Online]. Disponível: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>
- [23] Facebook, [Online]. Disponível: <https://pt-pt.facebook.com>
- [24] Flutter, [Online]. Disponível: <https://flutter.dev>
- [25] Dart, [Online]. Disponível: <https://dart.dev>
- [26] Xamarin, [Online]. Disponível: <https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin>
- [27] Microsoft, [Online]. Disponível: <https://www.microsoft.com/pt-pt>
- [28] Windows, [Online]. Disponível: <https://www.microsoft.com/pt-pt/windows/>
- [29] C#, [Online]. Disponível: <https://docs.microsoft.com/en-us/dotnet/csharp/>
- [30] .Net, [Online]. Disponível: <https://dotnet.microsoft.com>
- [31] Node.js, [Online]. Disponível: <https://nodejs.org/en/about/>

- [32] Python, [Online]. Disponível: <https://docs.python.org/pt-br/dev/faq/general.html>
- [33] C++, [Online]. Disponível: https://en.cppreference.com/w/cpp/language/basic_concepts
- [34] V8, [Online]. Disponível: <https://v8.dev>
- [35] Socket.IO, "*What Socket.IO is*", [Online]. Disponível: <https://socket.io/docs/v4/>
- [36] Mozilla, "*Express Web Framework (Node.js/JavaScript)*", [Online]. Disponível: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs
- [37] Socket.IO, "*Listening to events*", [Online]. Disponível: <https://socket.io/docs/v3/listening-to-events/>
- [38] Oracle, "*REST API Methods*", [Online]. Disponível: https://docs.oracle.com/en/cloud/saas/enterprise-performance-management-common/prest/rest_api_methods.html
- [39] AsyncStorage, [Online]. Disponível: <https://reactnative.dev/docs/asyncstorage>
- [40] Google Maps, [Online]. Disponível: <https://www.google.com/maps>