TÉCNICO LISBOA

App para Gestão de Frotas Aplicação Multiplataforma

Bruno Miguel Nascimento Carola

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Orientador: Prof. Alberto Manuel Ramos da Cunha

Júri

Presidente: Prof. José Luís Brinquete Borbinha

Orientador: Prof. Alberto Manuel Ramos da Cunha

Vogal: Prof. Renato Jorge Caleira Nunes

Outubro de 2019

Agradecimentos

Ao longo desta etapa da minha vida, o Mestrado em Engenharia Informática e de Computadores, fui sujeito a uma unidade curricular para a terminar a Dissertação, na qual tive o apoio de várias pessoas.

Primeiramente quero agradecer aos meus pais por todo o apoio, pelo financiamento inicial do curso, pela oportunidade que me deram de seguir os meus sonhos; pelas portas que me ajudaram a abrir para o meu futuro.

De seguida quero agradecer à Temic S.A. e a todos os colaboradores pela oportunidade que me deram na minha estreia no mundo profissional e posteriormente de finalizar o meu curso com a iniciativa desta tese.

Um especial agradecimento ao Coorientador João Domingos e ao Pedro Marques por toda a motivação, orientação, conselhos e por todas as discussões que houve em redor deste projeto. A toda a equipa de desenvolvimento pela espetacular integração a nível social.

Estou muito agradecido também ao Professor Alberto Cunha pela oportunidade que me deu de seguir em frente com esta dissertação, por todas as luzes que disponibilizou. Por ter aceitado trabalhar comigo e todo o feedback que me deu nas várias fases desta minha etapa.

Por fim devo um especial agradecimento á Margarida Almeida por todo o apoio, pela compreensão e paciência que teve ao longo deste ano e desta fase da minha vida. Muito obrigado Margarida Almeida.

Obrigado a todos!

Abstract

Technological evolution has been huge in the last decades, especially in the field of communication, more specifically in mobile technologies. Currently the market share is led by three mobile operating systems – Android, iOS and Windowns Phone – as well as a large number of downloadable mobile applications – 2.6 million for Android and 2.2 million for iOS – so there is a necessity to keep up with said evolution.

This document aims to showcase the development of a cross-platform mobile application focusing on Google's Android operating system and Apple's iOS. It's a car fleet management application called FleetMobile from Tecmic S.A ..

This application was only available for Android systems, so this document intends to show the entire process of migrating it to the iOS system and consequently to a multiplatform system, displaying the decision making process and choices made along the way, working simultaneously with the growth, needs and demands of the technological world and users of the FleetMobile app.

This paper addressed the need urgency to make code reusable and a way to facilitate future maintenance and upgrade actions evenly for both target systems of interest, using various techniques and technologies, from *Mvvm-Cross*, Xamarin to Android and iOS. Additionally, Signal-R technology was used to satisfy the requirement of having real-time information on some application features.

Keyword

Mobile Application, Fleet Management, Cross-Platform, iOS, Android, Design Pattern, Real Time Communications, Xamarin, Mvvm-Cross

Resumo

A evolução da tecnologia tem sido enorme nas últimas décadas principalmente a nível das tecnologias móveis. Atualmente existem três sistemas operativos moveis que lideram a cota do mercado – Android, iOS e Windows Phone – bem como um elevado número de aplicações móveis disponíveis para download -2.6 milhões para Android e 2.2 milhões para iOS - havendo assim uma necessidade de acompanhar esta evolução.

A redação deste documento visa mostrar o desenvolvimento de uma aplicação móvel multiplataformas com foco nos sistemas operativos Android da Google e iOS da Apple. Trata-se de uma aplicação de gestão de frotas automóveis denominada FleetMobile da empresa Tecmic S.A..

Esta aplicação encontrava-se disponível apenas para sistemas Android, pelo que este documento vem mostrar todo o processo de migração desta aplicação para o sistema iOS e, consequentemente, para um sistema multiplataforma, explicando todas as tomadas de decisão e escolhas efetuadas ao longo do percurso, trabalhando em simultâneo com o crescimento, as necessidades e as exigências do mundo tecnológico e dos utilizadores da aplicação FleetMobile.

Neste projeto endereçou-se a necessidade de tornar o código reutilizável de forma a facilitar futuras ações de manutenção e atualização de forma uniforme para os dois sistemas alvo de interesse, com o uso de várias técnicas e tecnologias, desde o *Mvvm-Cross*, o Xamarin para (Android e iOS). Adicionalmente usou-se a tecnologia Signal-R para satisfazer as várias necessidades de ter informação em tempo real em algumas funcionalidades da aplicação.

Palavra Chave

Aplicação Móvel, Gestão de Frota, Multiplataformas, iOS, Android, Padrão de Desenho, Comunicações Tempo Real, Xamarin, Mvvm-Cross

Conteúdo

Índice de Ilustrações

Acrónimos

API	Application Programming Interface
DAL	Data Access Layer
GPRS	General Packet Radio Service
GPS	Global Positioning System
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IDE	Integrated Development Environment
IoT	Internet of Things
IST	Instituto Superior Tecnico
JSON	JavaScript Object Notation
MVC	Model-View-Controller
MVP	Model-View-Presenter
MVVM	Model-View-ViewModel
OS	Operating System
PDA	Personal Digital Assistant
REST	Representational State Transfer
RPC	Remote Procedure Call
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SVN	Apache Subversion
TCP	Transmission Control Protocol
UDP	User Datagram Protoco
XML	eXtensible Markup Language

Capítulo 1

Introdução

1.1 Motivação

Nos últimos anos tem-se denotado um aumento exponencial na evolução da tecnologia mais concretamente em redor das tecnologias móveis. Na década de 2000 foram desenvolvidos os dois sistemas operativos atualmente mais usados em sistemas móveis, sendo estes o sistema Android e o iOS. Esta evolução tem inferido várias dificuldades no desenvolvimento de aplicações móveis, pois o mercado apresenta uma enorme necessidade de disponibilidade de aplicações para todas as plataformas.

A empresa Tecmic - Tecnologias De Microeletrónica – empresa multinacional portuguesa fundada em 1988, especializada na área de controlo e gestão de frotas – desenvolveu no ano de 2015 uma aplicação nativa em Android apelidada de FleetMobile que estende o produto iZiTraN Web. Esta permite visualizar, em tempo real, a posição precisa de veículos de uma frota num mapa e a informação dos mesmos, como dados pormenorizados dos tempos de condução, quilómetros percorridos, consumos de combustível, visualização de histórico de viagens do veículo, entre outros. A FleetMobile permite também aos operadores de frota contactar os seus condutores via mensagens de texto e receber alertas dos mais variados incidentes que possam surgir e tratar dos mesmos.

Com o passar do tempo os utilizadores desta aplicação tem vindo a exigir a disponibilidade da mesma para sistemas iOS. Tornar a aplicação nativa em Android numa aplicação multiplataforma, tendo em conta as várias funcionalidades e tecnologias, é um desafio.

1.2 Objetivos

Este projeto tem como foco satisfazer as exigências e as necessidades dos utilizadores da aplicação FleetMobile. Estudar a melhor metodologia para implementar de forma modular e interoperável esta aplicação, por forma a minimizar os custos futuros de manutenção e atualização em todas as plataformas necessárias (Android e iOS). Envolve assim o estudo de vários paradigmas peculiares como:

- Quais as melhores ferramentas para o desenvolvimento multiplataforma.
- O estudo do uso das guidelines de iOS às interfaces já desenvolvidas em Android.
- Qual a melhor tecnologia para comunicação em tempo real.
- Qual o melhor padrão de desenho e técnicas para tornar o código reutilizável e que facilite futuras ações de manutenção e atualização de forma uniforme para os dois sistemas alvo de interesse (Android e iOS).

1.3 Estrutura do Documento

Este documento encontra-se dividido em seis capítulos – incluindo o presente capitulo – mais uma secção de referências e anexos:

- **Capítulo 2 Estado de Arte –** neste capítulo encontra-se detalhado o Estado de Arte mostrando como está a FleetMobile Android e todas as tecnologias e plataformas já existentes que possibilitam o desenvolvimento de aplicações multiplataforma**.** Finalizando com um balanço de tópicos;
- **Capítulo 3 Proposta de Solução –** encontra-se descrito a proposta de solução e todas as escolhas feitas para prosseguir com o desenvolvimento deste projeto;
- **Capítulo 4 Implementação –** este capítulo descreve em pormenor o desenvolvimento de todo o projeto, começando por explicar as opções tecnológicas e a arquitetura do sistema informático existente, passando para o modelo de domínio e implementação das aplicações cliente e servidor;
- **Capítulo 5 Avaliação e Testes –** neste capítulo encontra-se detalhado todos os testes feitos. Com maior destaque para os testes unitários, de integração e de carga. Finalizando com uma discussão dos resultados dos mesmos;
- **Capítulo 6 Conclusão e Trabalhos Futuros –** por fim são apresentadas as conclusões e detalhadas considerações a ter em conta em trabalho futuro;

Capítulo 2

Estado de Arte

2.1 Trabalho Relacionado

A *FleetMobile* é uma aplicação profissional *Android* para gestão de frotas que permite o controlo e acompanhamento em tempo real de todos os veículos de uma frota. Esta aplicação é uma solução ideal para qualquer empresa, de qualquer dimensão, cuja atividade pressuponha a gestão e monitorização de frotas de veículos.

A aplicação *FleetMobile* permite ao operador de frota, com segurança, obter acesso ao estado dos veículos e realizar ações sobre os mesmos. Este pode visualizar em lista e em mapa o estado dos veículos (desligado, parado, em trânsito, em movimento), ter acesso a vários indicadores como localização, velocidade, quilómetros percorridos, consumos de combustível, entre outros. É também possível visualizar o histórico de viagens do veículo, o envio e receção de mensagens e os alarmes que podem ser desencadeados pelos veículos.

Foi desenhada de forma a ser modular, deste modo permitindo ser escalável com a adição e remoção de módulos, facilitando futuras manutenções, mudanças no código e agilizando a implementação do software, permitindo o desenvolvimento isolado nas diferentes partes da aplicação. Sendo que os módulos atualmente existentes são:

- *Real Time* – modulo onde é possível ver o mapa com a representação de todos os veículos;
- *Travels* – modulo que contem o histórico de viagens de todos os veículos do dia;
- *Alarms* – modulo que contem a listagem de todos os alarmes de cada veículo;
- *Messages* – modulo representativo da caixa de mensagens entre o operador e os seus veículos;

Este projeto consiste em tornar esta aplicação *Android* nativa numa aplicação multiplataforma. Na sequência deste processo terão de ser tomadas várias decisões e ultrapassar vários paradigmas.

2.2 *Frameworks* e Padrões de Desenho

Nesta secção será detalhado todas as *frameworks* e padrões de desenho do estado de arte para o desenvolvimento deste projeto.

2.2.1 Desenvolvimento separado - XCode

Atualmente, como já foi referido, a aplicação FleetMobile encontra-se desenvolvida em Android nativo usando a framework Android Studio. É possível seguir uma linha onde se mantém o trabalho já feito e desenvolve-se, de forma nativa, a versão em iOS. Obtendo duas aplicações isoladas.

Desde 2007, a Apple tem vindo a introduzir no mercado vários modelos de dispositivos móveis, desde *iPhones,* a *iPads,* a *iPods*, a *Apple Watches*, que fornecem um mercado potencial para aplicações móveis.

Em 2003, a Apple introduziu um ambiente de desenvolvimento integrado denominado *XCode*. Nas versões mais recentes, este ambiente vem com recursos modernos, mas poderosos, que fornecem um pacote completo. O *XCode* oferece aos desenvolvedores um modo de design de interfaces, um ambiente de desenvolvimento intuitivo e *User friendly*, como também um ambiente de testes de UI. O recurso mais poderoso do *XCode* é o *runtime*, que rastreia e alerta continuamente os programadores sobre erros de sintaxe, fornecendo recomendações e alertas de memória. Além disso, com a versão mais recente, o *XCode* fornece uma grande quantidade de extensões que podem ser integradas de forma fácil e inteligente ao *XCode*.

Como podemos ver na **Figura 1,** o *XCode* apresenta uma solução intuitiva e moderna para a construção das várias interfaces, baseando-se num pensamento de *Drag-and-Drop*, isto é, o pegar e largar componentes da biblioteca para a interface, tornado a aplicação apta para qualquer versão e modelo de dispositivos Apple.

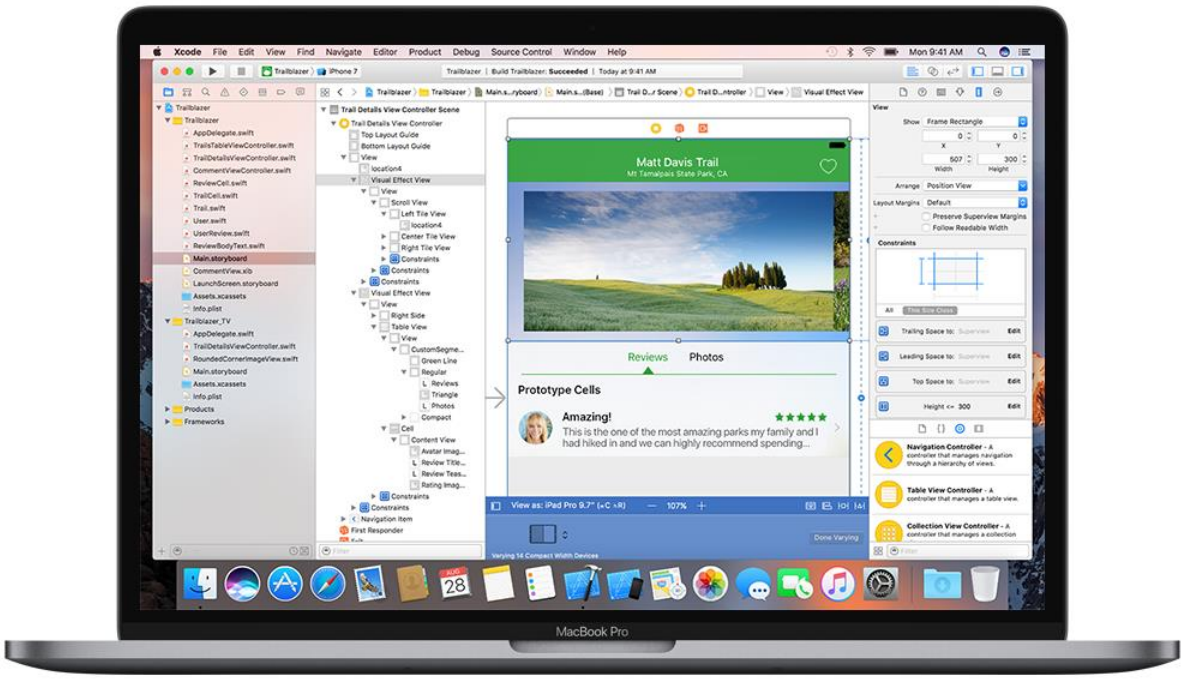


Figura 1 - Representação da Interface Builder do XCode:

Na **Figura 1,** o painel mais à esquerda representa o sistema de ficheiros da solução, seguido de um painel representativo da estrutura das várias componentes da interface. No painel central encontra-se representado o modo visual do desenho da interface. E o painel mais à direita é representativo da biblioteca de componentes e propriedades que serão Drag-and-Drop para o painel central, construindo assim a interface;

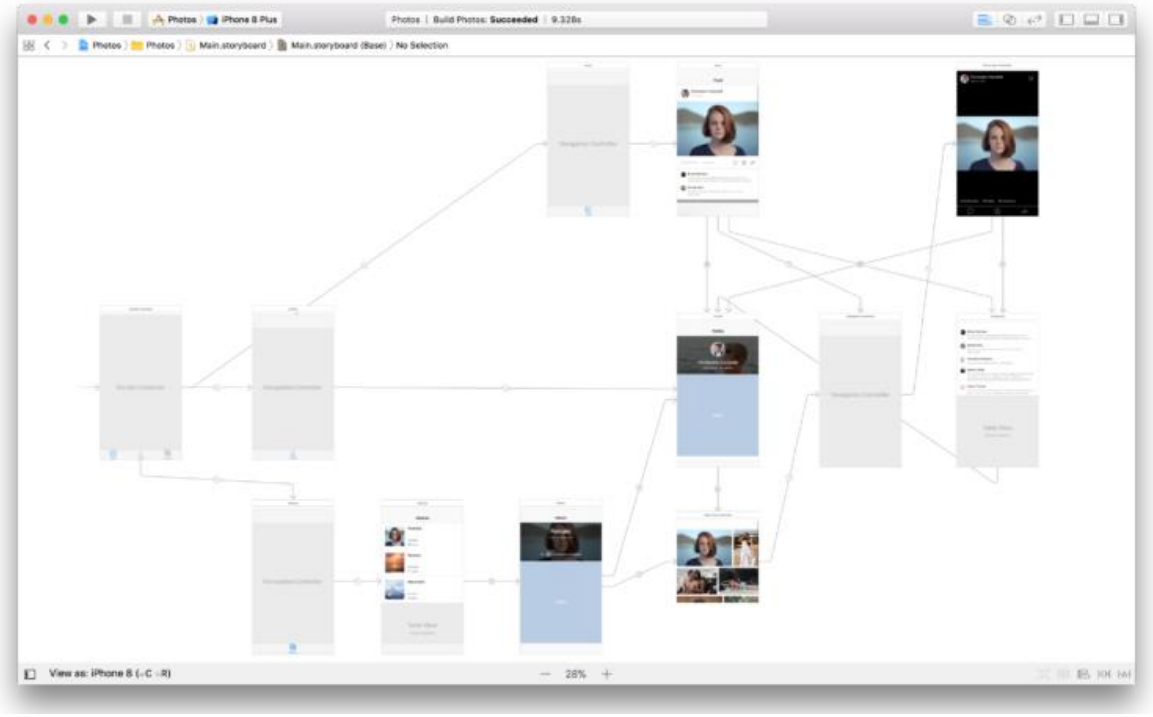


Figura 2 - Representação do Storyboard de uma aplicação de uma rede social para fotos. [7]

A *Interface Builder* do *XCode* apresenta também o conjunto das interfaces de todos os ecrãs de uma aplicação e a esquematização da navegação entre os mesmos de forma simples e intuitiva – A transição entre ecrãs a partir de cliques em elementos de interação, como botões ou elementos de listas – Através de traços que simulam esta mesma interação e por consequência a navegação para o ecrã de destino, como pode ser visto na **Figura 2**.

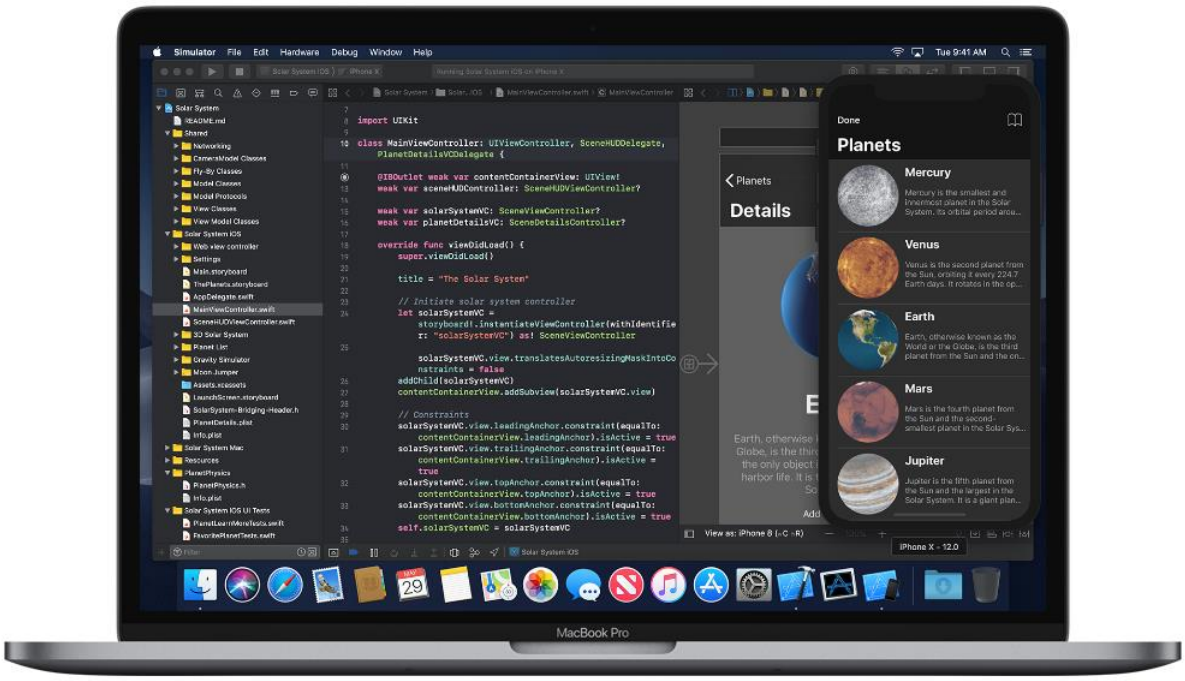


Figura 3 - Simulação de uma aplicação no iPhone X. [8]

Para além de uma forte componente de *design,* a Apple desenvolveu uma forte componente de testes e simulação. Nesta é possível testar e simular a aplicação desenvolvida em modo *debug* num simulador de qualquer dispositivo fabricado pela *Apple*, desde o *iPhone 4*, ao mais recente *iPhone*, passando pelos *Smartwatches* e pelos *iPods*. Como é possível ser visto na **Figura 3** um exemplo da simulação de uma aplicação num *iPhone X* com a versão *iOS 12.0***.**

O *XCode* suporta várias linguagens de programação e script, incluindo *C, C ++, Objective-C, Objective-C ++, Java, AppleScript, Python, Ruby, ResEdit e Swift*. Ainda assim, a Apple decidiu tornar o *Swift* como a principal linguagem de programação do *XCode*. [6] – O *Swift* é uma linguagem de programação criada pela Apple e foi desenvolvida a partir do *objetive-c*. No entanto, *Swift* é uma linguagem mais moderna e amigável que é desenhada para utilizadores e por utilizadores. Esta tem o poder de linguagens de programação de baixo nível como *C* ou *C ++* e a suavidade de linguagens de alto nível como *C #* ou *JavaScript*. Além disso, o *Swift* é leve e vem com uma biblioteca pré-definida bastante completa.

"Swift is a powerful and intuitive programming language for macOS, iOS, watchOS and tvOS. Writing Swift code is interactive and fun, the syntax is concise yet expressive, and Swift includes modern features developers love. Swift code is safe by design, yet also produces software that runs lightning-fast." – Afirmou a Apple em 2018.

O *XCode,* usando a linguagem de programação *Swift,* aparenta ser uma solução ótima para o desenvolvimento de aplicações *iOS* nativas, certamente terá todo o suporte tanto da marca como da comunidade de utilizadores.

2.2.2 Xamarin

Em alternativa ao desenvolvimento nativo, seria ter uma solução que conseguisse trabalhar com todas as possíveis plataformas, desde smart tvs, a smartwatchs, a tablets, a sistemas Android ou iOS. Esta secção apresenta uma solução multiplataforma desenvolvida pela Microsoft em 2011.

Xamarin é um *framework* multiplataforma que apresenta muitos recursos para construir aplicações móveis. Os criadores do *Xamarin* afirmam que o que é usual fazer com *Java, Objective-C* ou *Swift*, é também completamente exequível em *C #*. Utilizando todos os recursos do C # e as suas bibliotecas *.Net*, é possível criar aplicações nativas para *Android*, *iOS* e *Windows*.

Esta *framework* combina todo o poder das plataformas nativas acrescentando vários recursos poderosos por si mesma. Usando a linguagem C #, API e as estruturas de dados, em média, é possível desenvolver código onde 90% é compartilhado por todas as principais plataformas e com as interfaces do *Xamarin.Forms*, é possível compartilhar aproximadamente 100%. [9]

No *Xamarin* existe várias metodologias que podem ser usadas para compartilhar código de projetos multiplataforma como .Net Standard Libraries, Shared Projects e Portable Class Libraries, mais conhecidas com PCL, que serão detalhadas de seguida. [10]

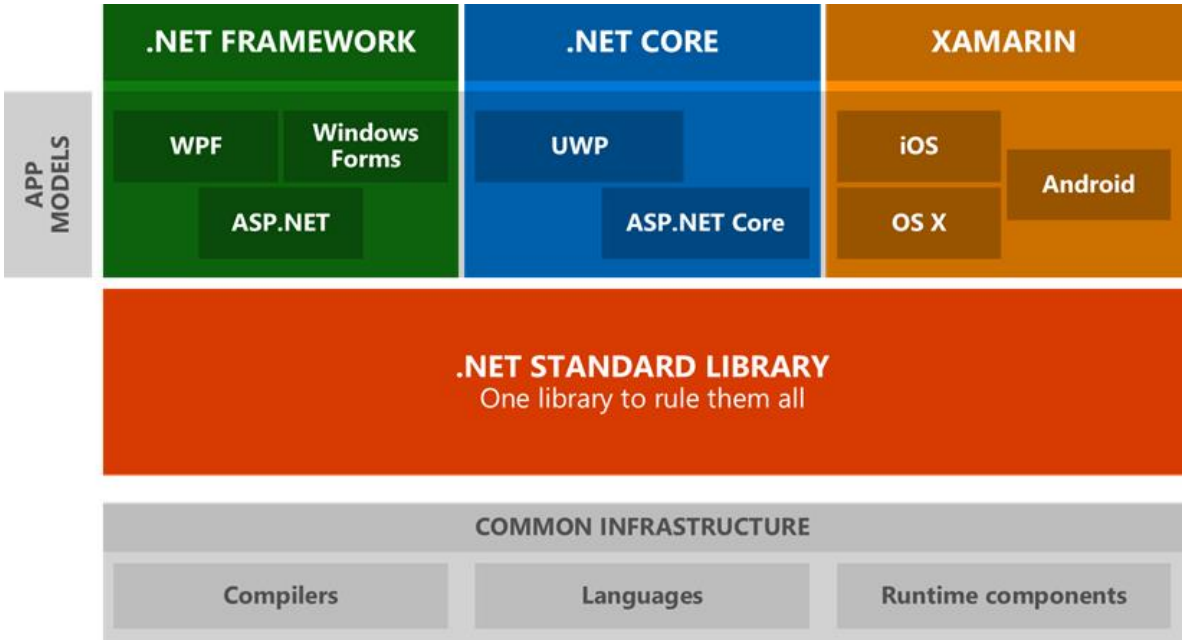


Figura 4 - Arquitetura .NET Standard Library. [10]

- *.NET Standard Libraries* – Na **Figura 4** podemos ver a Arquitetura destas, que fornecem um conjunto bem definido de bibliotecas de classes base que podem ser referenciadas em diferentes tipos de projetos, incluindo projetos multiplataforma, como *Xamarin.Android* e

Xamarin.iOS. O *.NET Standard 2.0* é recomendado pois apresenta compatibilidade máxima com o código existente da *.NET Framework*.

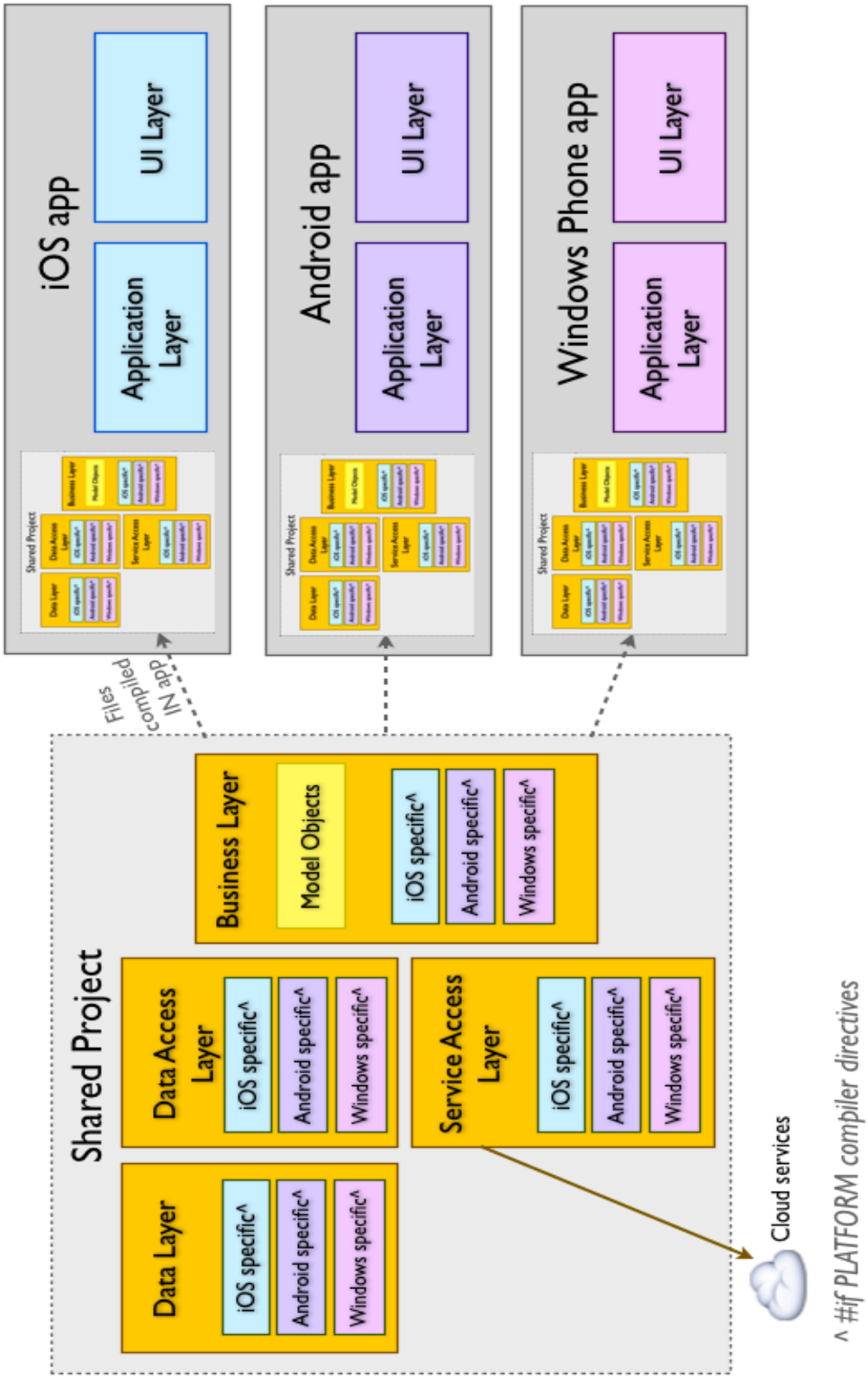


Figura 5 - Arquitetura Shared Projects. [10]

- *Shared Projects* – estas permitem que se escreva um código comum que é referenciado por diversos projetos de aplicações diferentes. O código é compilado como parte de cada projeto de referência e pode incluir diretivas do compilador para ajudar a incorporar a funcionalidade específica da plataforma na base do código compartilhada. Como está detalhado na **Figura 5.**

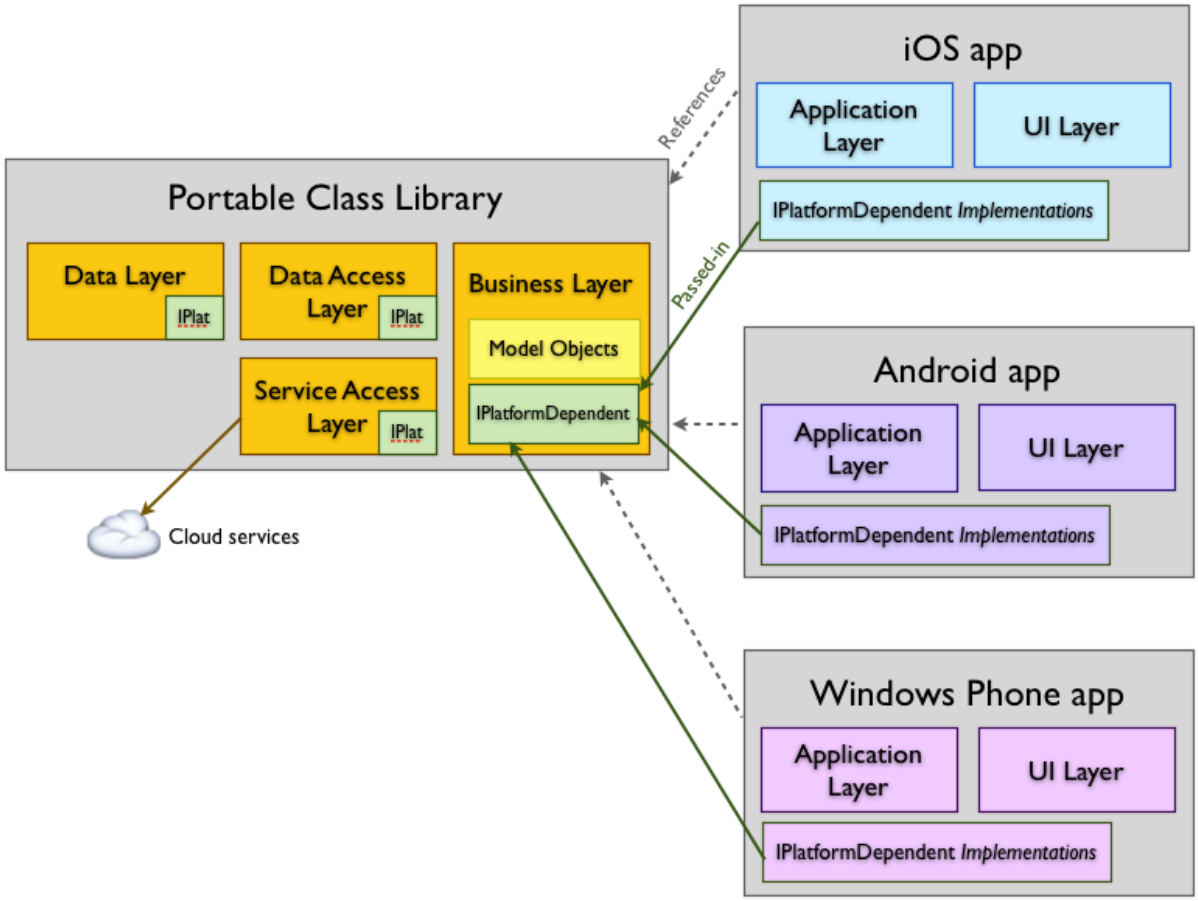


Figura 6 - Arquitetura Portable Class Library. [10]

- *PCL - Portable Class Library* – Quando é criado um *Application Project* ou um *Library Project*, a *DLL* resultante está restrita à plataforma específica para a qual foi criada. É possível também escolher uma combinação de plataformas nas quais se pretende que o código seja corrido. Como está detalhado na **Figura 6**. No entanto estas bibliotecas encontram-se obsoletas nas versões mais recentes do *Visual Studio*.

Esta *framework* apresenta várias componentes e opções de desenvolvimento de uma aplicação multiplataforma, que serão descritas de seguida.

Xamarin.Android permite que um programador de aplicações *Android* nativas use os mesmas componentes e comandos que usaria em *Java*, exceto com a flexibilidade e elegância da linguagem *C#*, tem também o complemento e a flexibilidade de usar as bibliotecas *.Net* e o uso de dois tipos de *IDEs*, o *Xamarin Studio* e o *Visual Studio*. A componente *Xamarin.Android* suporta também milhares de *APIs* o que simplifica o desenvolvimento de uma aplicação móvel multiplataforma sem perder desempenho comparativamente ao desenvolvimento nativo.

As várias interfaces podem ser criadas em ficheiros *XML*, à semelhança do *Android Studio,* ou programaticamente, inicializando componentes em código. Como alternativa, esta componente tem um modo *Designer* que permite criar e modificar *Layouts* visualmente, ao utilizar a técnica de drag-and-drop que consiste simplesmente em arrastar e largar no *Layout*, à semelhança do *XCode* anteriormente referido.

O *Xamarin* apresenta também uma componente denominada *Xamarin Android Player*, que é um simulador de um dispositivo Android que tem disponível todas as versões de Android e vários tipos de ecrãs para testar aplicações de forma fluida e intuitiva.

Xamarin.*iOS* permite construir aplicações *iOS* nativas usando as mesmos componentes e comandos de interface que são usadas no desenvolvimento em *XCode* usando *Objective-C* ou o *Swift*. Esta componente fornece a flexibilidade e elegância de uma linguagem moderna *C#* e o poder das bibliotecas *.Net*. Ou seja, agora com o *C#*, é possível fazer aplicações móveis nativas como quando se desenvolve em *Objective-C* ou *Swift*.

O *Xamarin.iOS* por poder ser utilizado tanto no *Xamarin Studio* como no *Visual Studio*, permite que seja possível desenvolver as aplicações para dispositivos iOS em sistemas Windows. Assim as aplicações iOS podem ser escritas e testadas no Visual Studio, com um Mac ligado em rede fornecendo serviços de *build e deployment*. O desenvolvimento de aplicações iOS no *Visual Studio* oferece vários benefícios enumerados de seguida:

- Torna possível criar projetos multiplataforma.
- Permite usar várias ferramentas do *Visual Studio*, prediletas pelos utilizadores como o *ReShaper* e o *Team Foundation Server*, que tornam o desenvolvimento de aplicações mais rápida e cuidada.
- O *Visual Studio* apresenta um modo de designer para o *Storybord iOS*, onde se pode simplesmente fazer o *Drag-and-Drop* de componentes para a interface, á semelhança do *XCode*.
- Os ficheiros de interface criados no *XCode* (xib e storyboard) são completamente compatíveis e facilmente importáveis para o *Visual Studio*.
- Fornece a possibilidade de se trabalhar num *IDE* familiar sem perder qualquer vantagem do trabalhar nativamente no *XCode.*

No *Xamarin.iOS* é possível criar os controlos programaticamente sem usar o Storyboard, o que ajuda quando as aplicações exigem que tudo esteja dinâmico.

NuGet é um *Package Manager* bastante popular na comunidade de desenvolvedores *C#* que está disponível tanto no *Xamarin Studio* como no *Visual Studio.* Este permite que os programadores criem, compartilhem e consumam códigos e bibliotecas *.Net* do projeto e para o projeto, fornecendo todas as ferramentas necessárias para cada uma dessas funções.

Mvvm-Cross é uma *framework* que foi desenvolvido pelo Stuart Lodge. O *Mvvm-Cross* é baseado no padrão de desenho *MVVM* (Model, View e View Model) que permite um aumento da capacidade de reutilização de código em todas as plataformas em que a aplicação irá ser desenvolvida.

O padrão de desenho *MVVM* separa a lógica de um objeto *View* em dois objetos separados, um chamado *View* e o outro é chamado *ViewModel*. O objeto *View* é responsável pela Interface do utilizador e o *ViewModel* são as classes principais onde toda a lógica comum a todas as plataformas é descrita.

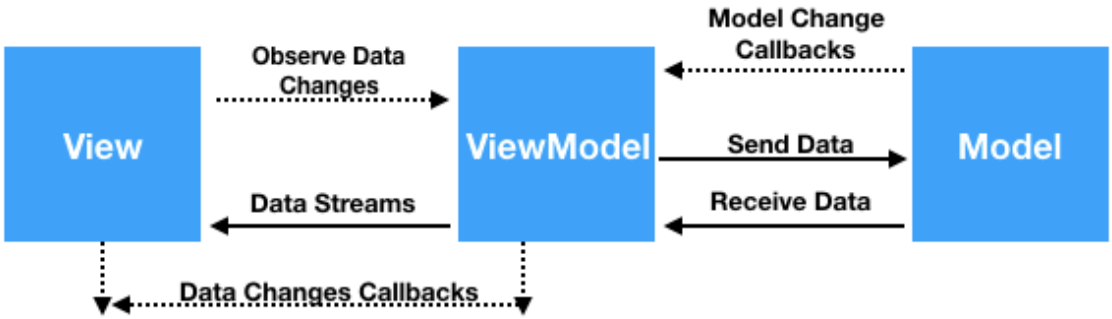


Figura 7 - Padrão de desenho MVVM. [11]

A *framework* do *Mvvm-Cross* usa o conceito de *PCL* como representação do *ViewModel*. Permite também o mapeamento de dados entre uma *View* e o *ViewModel*, pelo que é um recurso poderoso desta *framework*. A **Figura 7** esquematiza as várias interações entre as componentes deste padrão de desenho.

A lógica de navegação entre ecrãs das várias interfaces, um processo pouco simples, também é um aspeto para o qual o *Mvvm-Cross* tem solução. Esta funcionalidade é usualmente implementada no *ViewModel* que, a partir de um simples comando, consegue apresentar outro *ViewModel*, denominado *ShowViewModel<TObject>().*

O desenhar e desenvolver interfaces de todas as possíveis plataformas até agora só era possível separadamente, no entanto de seguida será detalhado uma solução que permite desenhar uma só interface que é mapeada para todas as possíveis plataformas e sistemas, sendo esta solução o uso do *Xamarin.Forms*.

Xamarin.Forms é uma componente que veio solucionar a criação das interfaces. Com o *Xamarin.Forms*, os desenvolvedores podem criar ecrãs de interface nativos, que podem ser compartilhados entre o *Android*, o *iOS*, o *Windows Phone* ou qualquer outra plataforma compativel.

O *Xamarin* promete até 100% de compartilhamento de código, o Xamarin.Forms, pode ser usado como a principal arma para isso, onde o compartilhamento de código é mais valorizado, inferindo um menor uso de funcionalidades específicas de cada plataforma, o que poderá ser um constrangimento.

Depois de desenhados os layouts é preciso mapear todos os elementos com o código *back-end* compartilhado e assim será possível ter aplicações *Android*, *iOS* e *Windows Phone* totalmente nativos. Durante o tempo de execução cada um dos elementos é mapeado para elementos de *UI* nativos específicos de cada plataforma, por exemplo uma componente de inserção de texto será mapeada para a *Caixa de Texto* no caso do *Windows-Phone*, o *UITextView* no caso dos sistemas *iOS* e o *EditText* no caso dos sistemas *Android*.

Como podemos ver na sequência dos vários pontos descritos, o *Xamarin* apresenta várias opções para o desenvolvimento de aplicações móveis, principalmente para o desenvolvimento multiplataforma.

2.2.3 PhoneGap

O *PhoneGap* é uma *framework* de desenvolvimento de aplicações móveis baseada num projeto *open source* (*Apache Cordova*). Esta *framework* permite aos desenvolvedores criar aplicações uma única vez em *HTML, CSS* e *JavaScript* e implantá-las em diferentes dispositivos móveis. O *PhoneGap* é uma ferramenta de desenvolvimento *Cloud-based* construída sobre a *framework*, que oferece um desenvolvimento de aplicações móveis baseados em nuvem sem a necessidade de *SDKs*, compiladores e *hardware*.

Devido aos padrões abertos *Web* do *PhoneGap*, os programadores com habilidades *Web* conseguem compreender e usar facilmente a *framework* e as funcionalidades existentes desta ferramenta. Ela também oferece um desenvolvimento multiplataforma com a liberdade de desenvolvimento personalizado. Por ser um serviço baseado em nuvem, todos as aplicações serão criadas com o *SDK* mais atual para cada plataforma de destino. O *PhoneGap Build* suporta múltiplas plataformas incluindo *Android*, *iOS*, *Windows Phone* e *webOS* com uma única base de código.

As aplicações resultantes são híbridas, no sentido de que não são nem aplicações móveis nativas nem puramente *web-based*. Em vez de utilizar a *UI* nativa de cada plataforma, toda a renderização das interfaces é feita por meio de *webviews*, por outro lado não se trata simplesmente de aplicações web pois estas aplicações têm acesso às *API's* nativas de cada dispositivo.

O *PhoneGap*, agora propriedade da *Adobe*, é totalmente gratuita, o que explica de alguma forma a sua popularidade. A *Adobe* também está a trabalhar numa versão empresarial do *PhoneGap*, que está atualmente numa versão beta. Esta versão possui recursos de marketing por meio do *Marketing Cloud* da *Adobe*, portanto quando for lançada provavelmente será monetizada.

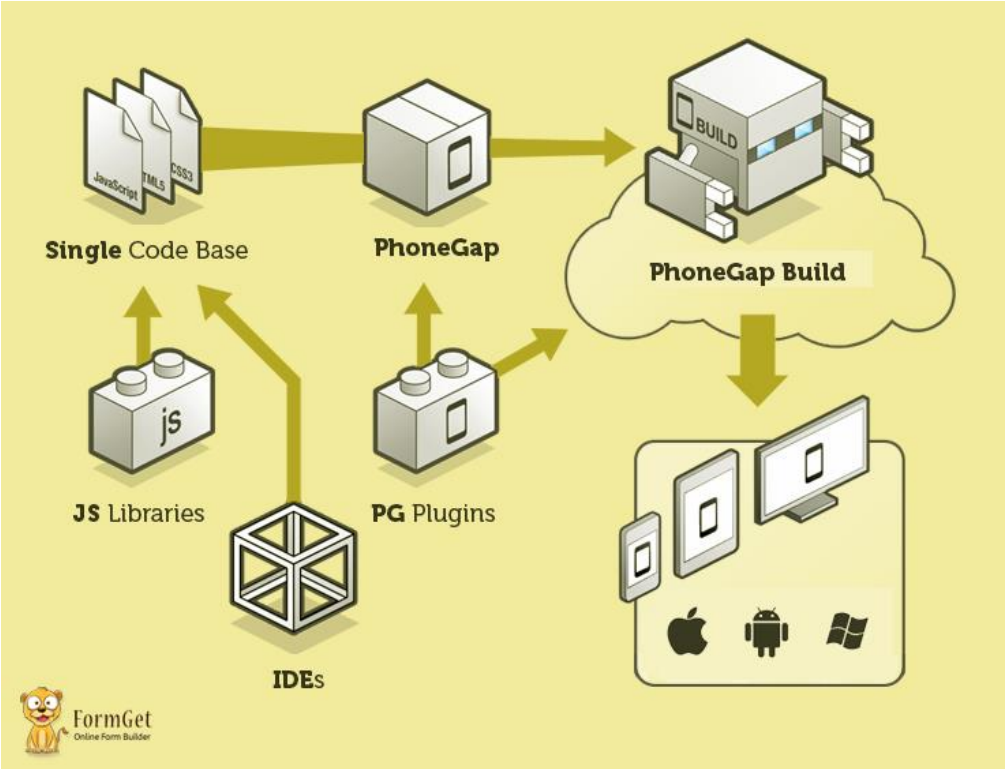


Figura 8 - Arquitetura PhoneGap. [12]

Na **Figura 8** pode ser visto a estrutura da arquitetura da ferramenta de desenvolvimento *PhoneGap.* Com o uso de um IDE e bibliotecas JS, é produzido um código base único, o qual é trabalhado pela framework PhoneGap e com a ajuda plugins é compilado e disponibilizado para todas as plataformas, desde sistemas iOS, Android a WindowsPhone.

Em suma esta *framework* apresenta as seguintes características/funcionalidades:

- Criação de aplicações para várias plataformas com uma base única de código;
- Desenvolvimento de aplicações em nuvem;
- *Open source software*;
- As aplicações criadas nesta *framework* não necessitam de manutenção a nível dos *SDKs*;
- Enorme número de plugins;
- Acesso a várias bibliotecas de JavaScript;
- *APIs* nativas o que torna as aplicações tão robustas com as nativas;
- Acesso a várias ferramentas *Third party* feitas pela comunidade e outras entidades;
- Garantia de 100% de compartilhamento de código;

2.2.4 React Native Framework

O engenheiro de software mobile estava restrito a certas linguagens, o que tornava os requisitos de entrada para se tornar um desenvolvedor de aplicativos nativo bastante limitado. As duas maiores plataformas móveis, iOS e Android, competem entre si desde o final dos anos 2000 e os desenvolvedores lutam para garantir que as suas aplicações recebam visibilidade nas duas plataformas. Com aplicações iOS programadas com *Objective-C*, e agora *Swift*, e aplicações Android escritas em Java ou Kotlin, pode levar aos desenvolvedores a despender de uma quantidade ineficiente de tempo para reescrever as suas aplicações numa linguagem totalmente diferente que também pode se comportar de forma significativamente diferente. Idealmente, um desenvolvedor gostaria de poder escrever e manter uma única aplicação para uma plataforma universal.[17]

O fundador do Facebook, Mark Zuckerber, começou uma entrevista em 2012 com *"Our biggest mistake was betting too much on HTML5". A*quando da discussão sobre a estratégia do Facebook para plataformas moveis este passou a usar uma aplicação nativa. Ficou claro que o Facebook se sentia restrito pelos recursos de uma aplicação híbrida, foi então que eles começaram o desenvolvimento da sua própria estrutura nativa, o que lhes permitiu utilizar totalmente todas as funcionalidades do dispositivo móvel. Essa estrutura nativa mais tarde ficou conhecida como *React Native*.

Arquitetura React - React e React Native usam ambos uma arquitetura que segue uma metodologia de fluxo para tratar da passagem de todos os dados na aplicação, denominada *Flux*. Esta foi criada pelo Facebook como alternativa ao convencional modelo *MVC – Model-View-Controller* [Secção 2.4.1]. A maior diferença do *Flux* para o *MVC*, é que o primeiro usa um modelo unidirecional na passagem de dados entre as várias camadas tornando mais fácil a identificação de erros – como está representado na **Figura 9**.

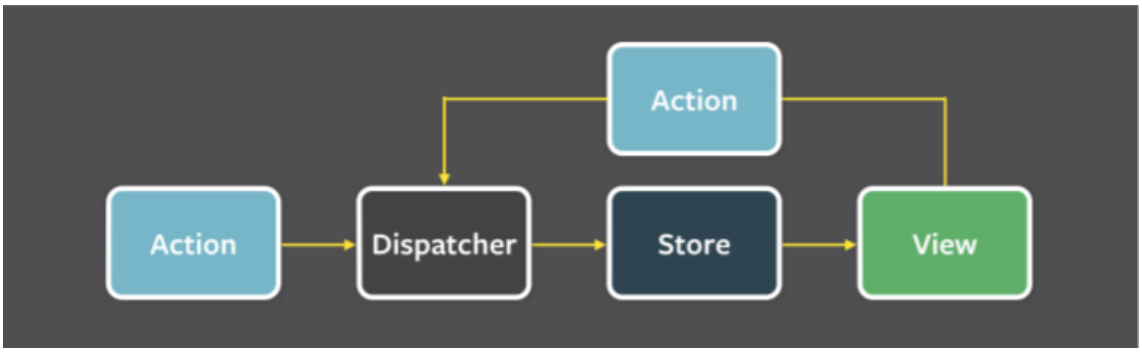


Figura 9 - Fluxo de informação numa aplicação React que usa Flux Architecture para construir interfaces;

A arquitetura *Flux* veio reduzir a complexidade de passagem de dados em comparação com o *MVC* por não usar ligações bidirecionais de dados e por ser obrigatório que todas a ações passem pelo *Dispatcher* centralizado – como encontra-se representado com a ligação de *Action* para *Dispatcher* na **Figura 9**. O *Dispatcher* trata de todas as alterações das diferentes fontes de dados procedendo á

atualização da *Store* – camada de dados da aplicação – por consequência a *View* – camada da vista – será atualizada e desenhada novamente de acordo com as novas alterações.

Desenvolvimento React – React Native usa muito uma logica de desenvolvimento Web, utilizado logica de HTML, a linguagem é Javascript e apresenta uma componente de estilos idêntica à do CSS. React Native, ao contrário do React, não usa as tags standard do HTML, em vez disso, criou componentes específicos semelhantes aos do HTML. A componente "Visualizar" por exemplo é muito semelhante à tag " ", pois ambos atuam essencialmente como contentor. O React Native também não usa ficheiros nem blocos individuais CSS para estilizar elementos, em vez disso, usa uma classe chamada "*StyleSheet*". Essa classe pode ser importada do React Native e cria um objeto que se assemelha com um objeto JSON, que pode ser utilizado na componente *Style* dos vários elementos.

Para resumir, o React Native permite que um desenvolvedor crie uma aplicação de forma simples. É uma forma eficiente de criar uma aplicação que pode ser mostrada diretamente no ecrã de um dispositivo móvel. Isso permite que o desenvolvedor veja as suas alterações atualizadas em tempo real, o que implica que as melhorias na interface e na experiência do utilizador possam ser testadas efetivamente com outras pessoas.

Um desenvolvedor também pode usar o React Native como uma extensão de um projeto nativo existente, facilitando o suporte a várias plataformas com uma única base de código. As ferramentas do React Native permitem que recursos específicos de uma plataforma sejam isolados da outra plataforma não interferindo entre si – Android e iOS.

2.3 Tecnologias para Comunicação em Tempo Real

A aplicação *FleetMobile* dispõe de um conjunto vasto de funcionalidades e dados em tempo real sobre a frota, como indicadores de localização, velocidade, estado atual das viaturas (desligado, parado, em trânsito, em movimento), quilómetros percorridos, entre outros.

No que diz respeito à comunicação, será utilizada a arquitetura *Representational State Transfer* (*REST*). Para além de ter sido uma decisão da equipa de desenvolvimento da empresa, é simples de entender e pode ser adotada em praticamente qualquer cliente ou servidor com suporte a *HTTP*/*HTTPS*. Os programadores que a utilizam citam como principais vantagens a facilidade no desenvolvimento, o aproveitamento da infraestrutura web existente e um esforço de aprendizagem pequeno. Sendo a maior vantagem da arquitetura *REST* a sua flexibilidade e o facto de dar a possibilidade ao programador de optar pelo formato mais adequado dos seus recursos – *XML, JSON* ou mesmo *HTML* – de acordo com as necessidades específicas. [4]

Para implementar as funcionalidades e os indicadores desta aplicação existe a necessidade de usar mecanismos de comunicação em tempo real onde foram escolhidas as seguintes opções.

2.3.1 SignalR

O *SignalR* simplifica o processo de adicionar funcionalidades de comunicação em tempo real à *framework ASP.NET* adotada nos *Web Servises* já existentes.

ASP.NET SignalR é uma biblioteca para programadores *ASP.NET* que simplifica o processo de adicionar funcionalidades de tempo real a aplicações. Estas funcionalidades consistem na aptidão de ter disponível o conteúdo resultante da execução do código do lado do servidor ligado aos clientes instantaneamente, ao invés do servidor esperar por pedidos recorrentes de clientes para obterem nova informação.

Esta tecnologia faz de forma automática a seleção do melhor método de transporte a partir da análise da conexão com o cliente, podendo optar entre:

- ***WebSockets*** – este método baseia-se numa comunicação *TCP*/*IP* de alto desempenho entre o servidor *Web* e o cliente. Neste modo a comunicação é bidirecional através do uso de uma ligação persistente entre os envolvidos;
- ***Long Pooling*** – este método é uma forma otimizada de *pooling* ao servidor *Web* para consulta de dados;
- ***Pooling*** – consiste numa comunicação normal com o servidor *Web* validando de tempos em tempos se há novõs pedidos para o cliente;

O *SignalR* contém dois modelos de comunicação entre clientes e servidores são estes o modelo de comunicação por **Ligações Persistentes** e o modelo por **Hubs** que serão detalhados de seguida.

- A **ligação persistente** dá ao programador acesso direto ao protocolo de comunicação de baixo nível que o *SignalR* expõe. Este modelo é utilizado principalmente quando se quer ter mais controlo sobre a transmissão de dados dando a possibilidade de se criar protocolos personalizados e técnicas como criptografia e compactação. Este modelo de comunicação é mais familiar para os programadores que utilizam *APIs* baseadas em ligação, como o *Windows Communication Foundation* (*WCF*).

- Já com o **modelo de comunicação por *Hub***, o programador tem um acesso de mais de alto nível sobre a *API* de ligação permitindo que o cliente e servidor invoquem os métodos diretamente. Este modelo de comunicação é mais natural para os programadores que utilizam *APIs* baseadas em invocação remota, como *.NET Remoting*. [4]

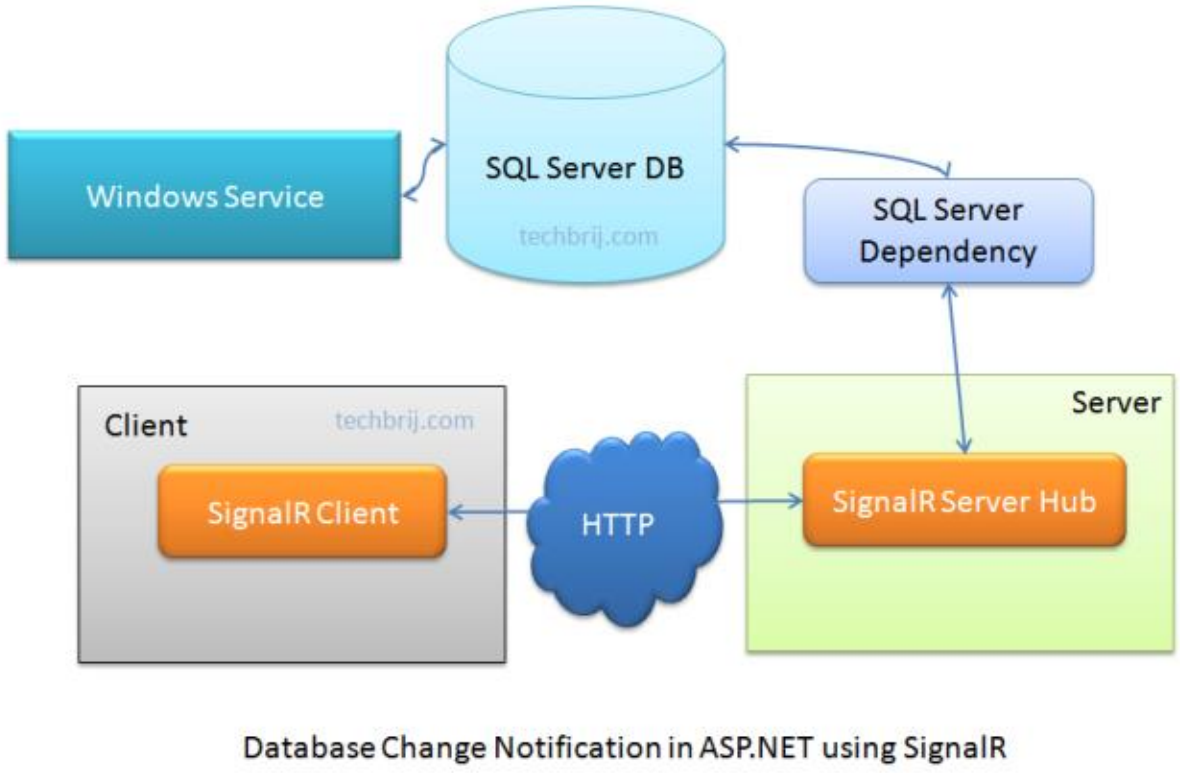


Figura 10 - Fluxo de notificação usando SignalR. [15]

Na **Figura 10** é detalhado o fluxo de notificação em tempo real de alterações a dados em base de dados. Este fluxo começa por haver uma alteração na base de dados, a qual é identificada pelo *SignalR Server Hub* e este notifica, neste caso, por *HTTP* os vários clientes. Assim, todos os clientes têm todos os indicadores e serviços atualizados em tempo real.

2.4 Padrões de desenho

Nesta secção irão ser discutidos quais os melhores padrões desenho tendo em conta dois principais objetivos:

- Melhores padrões de desenho para tornar código reutilizável;
- Melhores padrões de desenho que permitam futuras ações de manutenção e *updates* de forma uniforme sobre os dois sistemas alvo de interesse (*Android* e *iOS*);

A arquitetura de um sistema tem diversos elementos como: elementos utilitários, elementos de interação, elementos que fazem parte do domínio do problema, elementos de conexão, elementos de persistência, entre outros. É sempre interessante entender as características básicas de cada um dos estilos e escolher ou combinar aqueles que atendem melhor às necessidades de um projeto específico.

Para o desenvolvimento desta aplicação foram selecionados dois padrões de desenho, o já referido *MVVM* e o *MVC*.

2.4.1 MVC

Atualmente existe várias aplicações que usam a arquitetura *MVC* – *Model-View-Controller* – incluindo a aplicação *FleetMobile* nativa *Android*. Esta tem uma arquitetura composta pelas três camadas do modelo *MVC*, que determinam uma separação lógica da responsabilidade de processamento e tratamento de informação. [4]

Este padrão de desenho, se cada uma destas camadas forem implementadas corretamente, consegue proporcionar aos programadores uma manutenção mais fácil e um possível reaproveitamento de classes e partes do projeto para projetos futuros.

A utilização do padrão *MVC* traz como principal vantagem isolar as regras de negócios da lógica de apresentação, a interface. Isto possibilita a existência de várias interfaces que podem ser modificadas sem que haja a necessidade da alteração das regras de negócios, proporcionando assim muito mais flexibilidade e oportunidades de reuso de classes. [13] [14]

A comunicação entre as interfaces e as regras de negócios é definida através de um controlador e é a existência deste controlador que torna possível a separação entre as camadas. Quando um evento é executado na interface gráfica, *View*, como um clique de um botão, a *interface* irá comunicar com o Controlador que por sua vez comunica com as regras de negócios. [14]

Na **Figura 11** estão representados os elementos e as respetivas interações deste padrão de desenho, passando de seguida para a respetiva explicação dos mesmos.

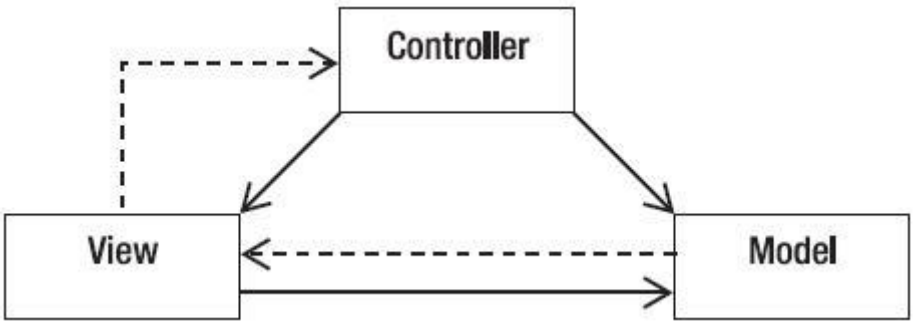


Figura 11 - Arquitetura do MVC.

O controlador (*Controller*) interpreta as interações do utilizador com a aplicação e mapeia essas ações em comandos que são enviados para o modelo (*Model*) e/ou para a interface (*View*) para efetuar a alteração apropriada. O modelo (*Model*) é a representação de um ou mais elementos de dados, responde a perguntas do seu estado e responde a instruções para mudar o mesmo. O modelo sabe o que a aplicação quer fazer e é a principal estrutura computacional da arquitetura, pois é nele que está modelado o problema. Por fim, a vista (*View*) é responsável por apresentar todas as informações ao utilizador através de uma combinação de componentes gráficas e textos. A vista não sabe nada sobre o que a aplicação está atualmente a processar, tudo o que esta realmente faz é receber instruções do controlador e informações do modelo procedendo para o mapeamento dos mesmos. A vista também comunica com o modelo e com o controlador para informar sobre o seu estado. [13] [14]

Entre as diversas vantagens do padrão *MVC* estão:

- Possibilidade de haver alteações da *GUI* (*Interface Gráfica do Utilizador*) ou do *Controller* sem alterar o nosso modelo.
- Reutilização da *GUI* para diferentes aplicações com pouco esforço.
- Facilidade na manutenção e adição de recursos.
- Reaproveitamento de código.
- Facilidade em manter o código sempre limpo.

2.4.2 MVVM

Como já foi referenciado na **secção 2.2.2** na parte do ***Mvvm-Cross*****,** o padrão de desenho *MVVM* traduz-se em *Model* – *View* – *ViewModel*.

O modelo (*Model*) é usualmente denominado como o objeto de domínio. O modelo representa os dados e / ou informações reais com os quais se está a lidar. A ideia chave para categorizar o modelo é que este contém as informações, mas não os comportamentos ou serviços que o manipulam. Não é responsável por formatar o texto na interface ou buscar uma lista de itens a um servidor remoto. A lógica de negócio normalmente é mantida separada do modelo e encapsulada noutras camadas que atuam sobre o modelo.

A vista (*View*) representa a camada de apresentação desta arquitetura muito resumidamente é responsável pela apresentação dos dados. Uma vista no *MVVM* contém comportamentos, eventos e ligações de dados que, em última análise, requerem conhecimento do modelo e do *ViewModel* subjacentes. Embora esses eventos e comportamentos possam ser mapeados para chamadas de métodos e comandos, a vista ainda é responsável por manipular os seus eventos e não deixar esta tarefa completamente para o *ViewModel*. A ideia chave da vista é que ela não é responsável por manter seu estado, em vez disso, encontra-se sincronizada com o *ViewModel*.

O *ViewModel* é uma peça-chave da tríade porque mantem as nuances da vista separadas do modelo. O *ViewModel* é responsável por expor métodos, comandos e outros pontos que ajudam a manter o estado da vista, manipulam o modelo como resultado de ações na vista e acionam eventos.

Em suma, a camada *Model* (Modelo) não conhece a *View* (Vista) e vice-versa, na verdade a *View* conhece a *ViewModel* e comunica com ela através de mecanismos de *binding*. E são estes mecanismos de *binding* avançados, eventos roteados e comandos roteados, que fazem do *MVVM* um padrão de desenho poderoso para construção de aplicações multiplataforma.

2.5 Balanço dos tópicos

Nesta secção será descrito o balanço do capítulo 2, os prós e contras de cada decisão. Quando se fala em desenvolvimento de software mobile, mais concretamente para *iOS* e para *Android*, a maioria pensa nos pros e dos contras da escolha *Objective-C vs Swift* e *Java vs Kotlin,* pois são as formas mais convencionais para programação mobile nativa. Contudo existem outros caminhos para o fazer que podem trazer mais performance.

Segue um quadro com a comparação entre o Xamarin, desenvolvimento nativo e desenvolvimento híbrido (PhoneGap, React Native). [18]

	Xamarin	Native	Hybrid
Tech stack	One tech stack, single codebase (C#, .Net framework + native libraries)	Different tech stacks for each platform.	One tech stack, single codebase (Javascript, HTML5, CSS)
Code sharing	Yes (up to 96% with Xamarin.Forms)	No, different code bases	Yes, 100%
UI/UX	Complete UI customization is possible for each platform (with Xamarin.iOS and Xamarin.Android)	Completely platform-specific UI	Common UI for all platforms (limited customization capabilities)
Performance	Good, close to native	Excellent	Medium - Poor
Hardware capabilities	High - Xamarin uses platform-specific APIs, and supports linking with native libraries	High - Native tools have complete support for system capabilities out of the box.	Medium - The capabilities can be accesses through third-party APIs and plugins, although there are some risks due to the poor quality and unreliability of most of these tools.
Time to market	With Xamarin.Forms the time to market is fast due to the limited customization and extensive code sharing. Xamarin.iOS and Xamarin.Android require slightly more time as the amount of custom code increases.	The time to market for iOS or Android native app might equal that of Xamarin.Forms or Hybrid tools. Yet, building apps for multiple platforms, will require you to either prolong time to market or increase the number of developers involved.	Hybrid solutions offer the fastest time to market thanks to the single code base and minimum customization. These tools are even used for prototyping and proof of concept projects.

Figura 12 - Xamarin vs Aplicações Nativas vs Aplicações Híbridas. [18]

As aplicações Híbridas apresentam o melhor nível de partilha de código, apresentam 100% enquanto que o *Xamarin* apresenta no máximo 96%. São também a solução que apresenta o melhor tempo de publicação das aplicações no mercado. No entanto, a nível de performance e compatibilidade de hardware ficam um pouco atras. O React Native sobressai um pouco destas desvantagens pois nos últimos anos tem melhorado bastante a sua performance, conseguindo chegar a valores de performance idênticos ao do desenvolvimento Nativo, como podemos ver na Figura 13.

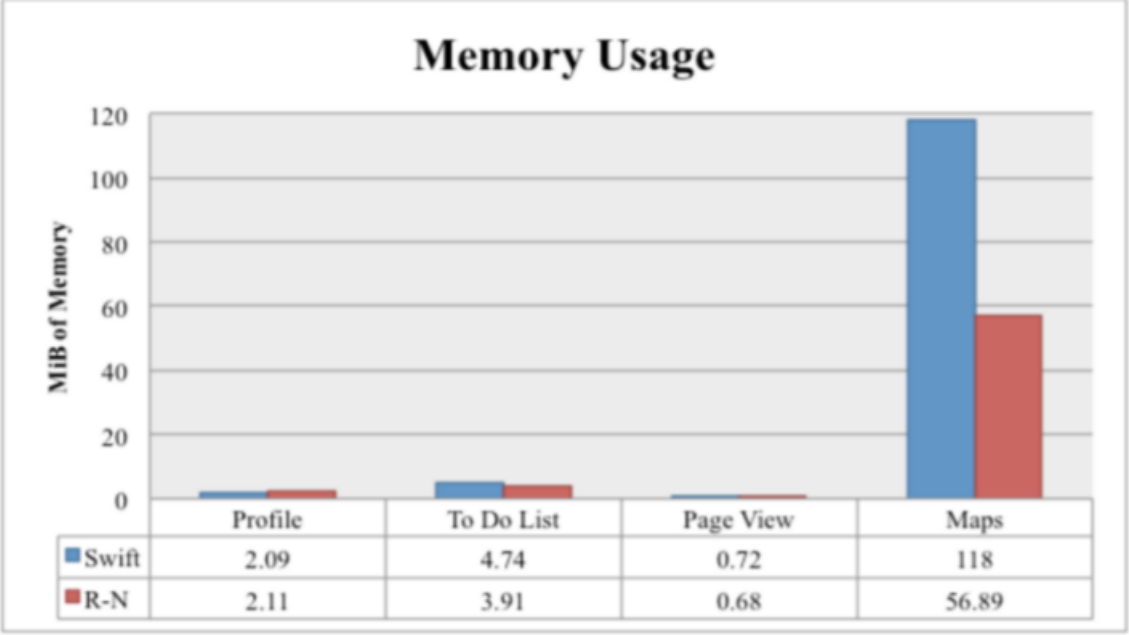


Figura 13 - Comparação de performance entre Nativo (Swift) e React Native (2016) [17]

O *Xamarin*, com a sua forte compatibilidade ao desenvolvimento multiplataforma juntamente com a liberdade na escolha do *IDE*, é uma das maiores alternativas. Esta ferramenta permite partilhar aproximadamente 96% do código sobre as todas as plataformas e apresenta uma comunidade que ronda os 1.4 milhões de desenvolvedores. É baseado em *.NET framework* e como já foi dito o *C#* é uma linguagem bastante matura e evoluída, podendo ser utilizada em paralelo com várias tecnologias uteis para o desenvolvimento, como o *LINQ*, *Lambdas* e programação *Assíncrona*.

Ao contrário das soluções convencionais para o desenvolvimento híbrido, baseadas em tecnologias web, uma aplicação multiplataforma criada usando o *Xamarin*, pode ainda ser classificada como nativa. A sua performance tem sido constantemente melhorada, de *update* para *update*, para se aproximar ao máximo do nativo, um pouco devido ao facto de ser open-source e por ter uma comunidade em constante crescimento.

Capítulo 3

Proposta de Solução

A solução passou em parte pela decisão da equipa de desenvolvimento da Tecmic, sendo que os pontos mais relevantes se encontram detalhados de seguida:

1. Esta aplicação será desenvolvida usando o *Xamarin* no *Visual Studio*. Isto com base na discussão da secção anterior como também pelos sistemas já existentes na empresa, todas as soluções dos projetos da *Tecmic* encontram-se desenvolvidas pelo *Visual Studio*.

2. Para desenhar as interfaces iOS será utilizado o *XCode*, as quais serão importadas no fim para a solução desenvolvida no *Visual Studio*, conseguindo ter as componentes nativas do *XCode* e as *Guidelines* da *Apple* na solução desenvolvida, tendo o melhor dos dois mundos.

3. Optou-se por usar o plugin do *Mvvm-Cross* disponível na componente *NuGget*, descrita na secção anterior, por consequência usar o padrão de desenho *MVVM*. Isto para tornar código reutilizável para outras aplicações e agilizar futuras ações de manutenção e atualização de forma uniforme para os dois sistemas alvo de interesse (*Android* e *iOS*).

4. Em termos da tecnologia de comunicação tempo real decidiu-se utilizar o *SignalR*. A opção do modelo de comunicação recaiu para a utilização por *Hubs*, devido à experiência adquirida da equipa de desenvolvimento e à sua utilização na maioria das aplicações, recomendada mesmo pela própria *Microsoft*.

Esta solução foi complementada ao longo do tempo com mais plugins e melhorias.

Capítulo 4

Implementação

Nesta secção será descrito todo o processo de desenvolvimento, começando por um resumo geral das tarefas e desafios que se pretendem cumprir e detalhando os principais paradigmas deste projeto.

4.1 Resumo Geral

Como foi referido na **secção 1**, este projeto tem como foco satisfazer as exigências e as necessidades dos utilizadores da aplicação FleetMobile e estudar a melhor metodologia para implementar de forma modular e interoperável esta aplicação, procurando minimizar os custos futuros de manutenção e atualização em todas as plataformas necessárias (Android e iOS) e envolvendo assim o estudo dos vários paradigmas, enumerados na **secção 1.2**.

Nesta secção será detalhado todo o desenvolvimento da aplicação. Começando pelo enquadramento tecnológico onde serão referidos os sistemas mínimos e de teste escolhidos para a disponibilização da aplicação, à parte de todos os sistemas envolventes da mesma. Na secção seguinte será detalhada toda a componente servidor usada pela aplicação e todo o seu modelo de domínio. Por fim será mostrada toda a aplicação desenvolvida e as tecnologias utilizadas.

4.2 Enquadramento Tecnológico

Entre os principais sistemas operativos para smartphones, a empresa optou que a aplicação móvel fosse desenvolvida para a plataforma Android com versões iguais ou superiores à versão KitKat 4.4.

Por outro lado, em relação ao sistema operativo iOS será disponibilizada em sistemas iguais ou superiores ao Iphone 5 e a versão mínima 10.3 do sistema, sendo, no entanto, sempre recomendado ter sempre a atualização mais recente. Foi assim escolhido devido à cota do mercado da versão 10.3 em janeiro de 2019 ser ainda de 4.17% sendo que a versão mais recente neste mesmo mês, iOS 12.1, ser de 67.24%. Juntando o facto dos dispositivos iOS existentes na empresa serem o modelo 5S, o 7 e o modelo X. [16]

4.2.1 Ferramentas de desenvolvimento

Para o desenvolvimento de toda a solução da aplicação FleetMobile foi usada a ferramenta Microsoft Visual Studio, à semelhança para os Web services desenvolvidos foi também utilizada a ferramenta Microsoft Visual Studio adotando a framework ASP.NET. A equipa de desenvolvimento foi novamente um fator decisivo na escolha da framework ASP.NET devido à enorme experiência já adquirida na área, uma vez que muitos dos produtos existentes na empresa são desenvolvidos através desta.

Em relação ao armazenamento e controlo de versões do código criado foi utilizado o controlo de versões SVN (Apache Subversion) da própria Tecmic S.A.. Este tipo de ferramentas é sempre útil no desenvolvimento de software, pois permite um armazenamento do código e para além disso permite um desenvolvimento paralelo com a restante equipa, reduzindo o tempo de resolução de conflitos no código pelos elementos da mesma. A maioria das implementações possibilita a divisão do projeto em várias linhas de desenvolvimento, que podem ser trabalhadas paralelamente, sem que uma interfira com a outra. Também existe um controlo de histórico que não só permite analisar o registo temporal dos desenvolvimentos, como também facilitar o resgate de versões mais antigas e estáveis. A maioria das implementações permitem analisar as alterações com detalhe, desde a primeira versão até a última. Tanto no sistema da Tecmic S.A., mas também na maioria dos sistemas SVN, estes permitem marcar a altura em que o ficheiro estava com uma versão estável, podendo ser facilmente recuperado posteriormente. [4]

4.2.2 Protocolos e Tecnologias de Comunicação

Como já foi referenciado na **secção 2.3**, foi usado o ***REST*** *(Representational State Transfer)* no que diz respeito à comunicação. REST é simples de entender e apresenta como uma das suas principais características a forte compatibilidade com qualquer cliente ou servidor que suporte *HTTP/HTTPS*, sendo a maior vantagem da arquitetura *REST* a sua flexibilidade e o facto de dar a possibilidade ao programador de optar pelo formato mais adequado de acordo com as necessidades específicas. Os formatos mais comuns são *JavaScript Object Notation (JSON), eXtensible Markup Language (XML)* e texto puro, mas em teoria qualquer formato pode ser usado [4]. O que leva a outra vantagem a nível de performance, tendo em conta que os Web services podem usar o formato JSON (que é o utilizado nesta solução) leva a um aumento de performance devido ao tamanho dos pacotes. Isto é, necessitam de **menor largura de banda**, que é bastante útil nas comunicações em dispositivos móveis. No entanto, é de realçar o crescimento da utilização dos protocolos binários, assunto muito debatido na temática *Internet of Things (IoT)*. [4]

Na aplicação móvel FleetMobile recorreu-se ao uso da *API* de comunicações disponibilizada no *NuGet* denominada *Microsoft.NET.Http* e nos *Web Services* utilizou o que já estava implementado, que usa a *framework ASP.NET Web API* para comunicar com a mesma. A *Web API* é uma das *frameworks* mais utilizadas na empresa, com ela torna-se mais fácil construir serviços *HTTP* que chegam a uma ampla gama de clientes, incluindo *Web browsers* e dispositivos móveis, permitindo

que os pedidos sejam independentes da plataforma, seja web seja qualquer plataforma Mobile [4]. Com a *Web API* pode negociar-se a representação de dados, ou seja, se o cliente está a solicitar dados a serem retornados como *JSON* ou *XML*, a *Web API* lida com o tipo de solicitação e retorna os dados de forma apropriada com base no tipo pretendido. O cliente pode assim fazer um pedido *GET, PUT, POST* ou *DELETE* e obter a resposta ao pedido *REST* que será transmitida via *HTTP/HTTPS*. [4]

Devido às necessidades dos vários utilizadores em ter informação em constante atualização sobre os veículos, mensagens, viagens e alarmes. Para estas mesmas funcionalidades em tempo real da aplicação, foi usado a *API* da *framework ASP.NET, Signal R* [15], que permite carregar informação do lado do servidor para o cliente sempre que haja uma alteração nos dados referentes aos indicadores em tempo real apresentados na aplicação como, todos os indicadores de velocidade, instantânea e média, como os indicadores de consumo do veículo, como o estado atual do veículo, em movimento, em trânsito e parado. Foi escolhida esta *framework* para reduzir o número de ligações e pedidos ao servidor, para ajudar a redigir como o servidor trata todos os pedidos e para garantir que a informação chega a tempo útil reduzindo todos os constrangimentos que pode haver em comunicações em tempo real.

Esta biblioteca tem código aberto, uma enorme facilidade na implementação de comunicação em tempo real e uma forma simples de tratar atualizações/notificações de forma assíncrona [4]. Existem algumas técnicas/tecnologias conhecidas de implementar uma comunicação em tempo real (*WebSockets, Long Polling*, entre outras), sendo que possuem uma complexidade ou limitação técnica. O *SignalR* propõe facilitar a sua implementação, uma vez que fica a cargo do próprio encontrar o transporte mais eficaz entre um servidor e cliente. Sempre que a tecnologia *WebSockets* estiver disponível é-lhe dada preferência, contudo se não existir esta disponibilidade o *SignalR* opta pelo *Long Polling* e as demais em sequência. [4][15]

A *framework SignalR*, como já foi referenciado na **secção 2.3.1**, apresenta várias possibilidades para criar e manter ligações cliente-servidor por *WebSockets,* onde a comunicação é feita por *TCP*/*IP* de alto desempenho entre o servidor *Web* e o cliente. Neste modo a comunicação é bidirecional através do uso de uma conexão persistente entre os envolvidos. Por outro lado, o *Pooling* é considerado uma comunicação normal com o servidor *Web* validando de tempos em tempos se há novos pedidos para o cliente. A técnica, que também existe no *SignalR*, denominada de *Long Polling*, consiste numa variação do conceito de *Polling*, porém cada pedido efetuado pelo cliente é guardado até que o servidor tenha algum dado para enviar. Após isto, a ligação é fechada e o cliente deve estabelecer uma nova ligação. A vantagem neste caso é que não há pedidos desperdiçadas enquanto o servidor não tem novos dados, uma vez que a ligação entre o cliente e o servidor se mantém aberta por um determinado período, até que seja enviada uma resposta através deste último. Como se disse anteriormente, fica a cargo do *SignalR* encontrar o transporte mais eficaz entre um servidor e cliente, dando sempre preferência por *WebSockets*, seguindo-se pelo *Long Polling* e os demais em sequência, facilitando assim a implementação da comunicação em tempo real. [4][15]

4.3 Arquitetura da Aplicação e Infraestrutura

Nesta secção será descrita toda a arquitetura da aplicação e infraestrutura, tanto o modelo dados como todo o conjunto dos seus ecrãs e funcionalidades.

4.3.1 Infraestrutura do Sistema

Como já foi referido este projeto surgiu da necessidade de existência de uma versão para sistemas iOS da aplicação Fleetmobile. Como tal, esta foi naturalmente integrada na infraestrutura do sistema já existente. Segue-se a **Figura 14** que vem descrever essa mesma infraestrutura denominada iZiTraN.

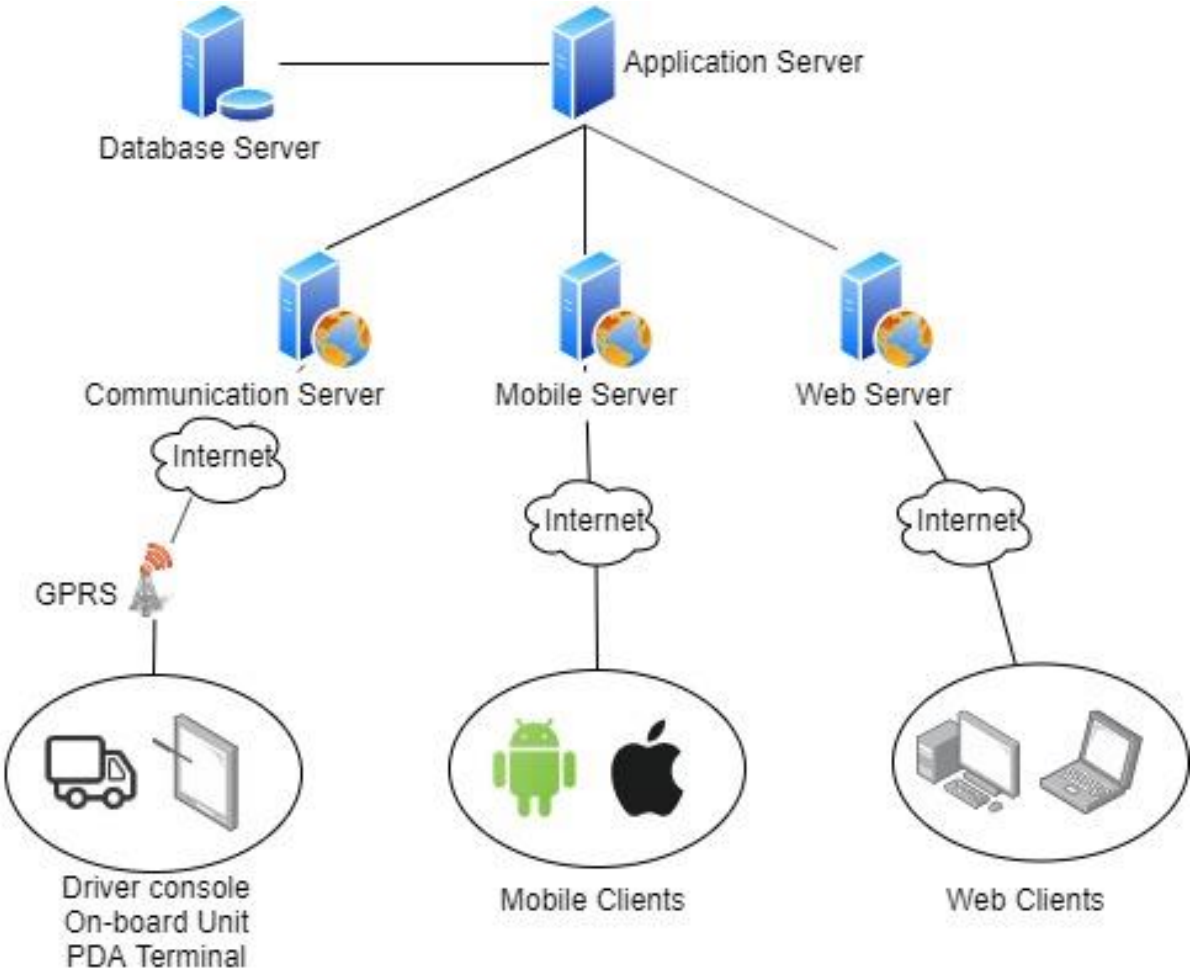


Figura 14 - Arquitetura da infraestrura do sistema iZiTraN;

A infraestrutura iZiTraN é constituída pelos vários dispositivos e equipamentos Tecmic presentes nos veículos da frota. No canto inferior esquerdo da **Figura 14** estão representados os equipamentos Tecmic que permitem deduzir e retirar toda a informação necessária referente a cada

veículo, estes equipamentos variam desde as consolas dos condutores (**Figura 15 e Figura 16**), terminais *PDA* (**Figura 17**) e as unidades a bordo colocadas no veículo. Estes dispositivos retiram do campo toda a informação:

- Geográfica;
- Velocidade;
- Consumo;
- Estado do veículo;
- Informação de operações;
- Informação de viagens;
- Eventos;
- Entre outros dados sobre o veículo e o estado de negócio;

É usado o serviço *GPRS* sobre as redes *GSM* existentes, cuja ligação à Internet possibilita o encaminhamento destes mesmo dados ao servidor de comunicações (*Communication Server*), para que este trate das comunicações recebidas e as envie pela rede interna da empresa ao servidor aplicacional (*Application Server*). O *Application Server* assenta numa *Service-Oriented Architecture* (*SOA*) que contém lógica de negócio distribuída, sendo responsável por centralizar toda a informação recebida, desde equipamentos Tecmic. terminais *PDA*, como outras integrações.

Figura 15 - Consola gráfica do condutor;

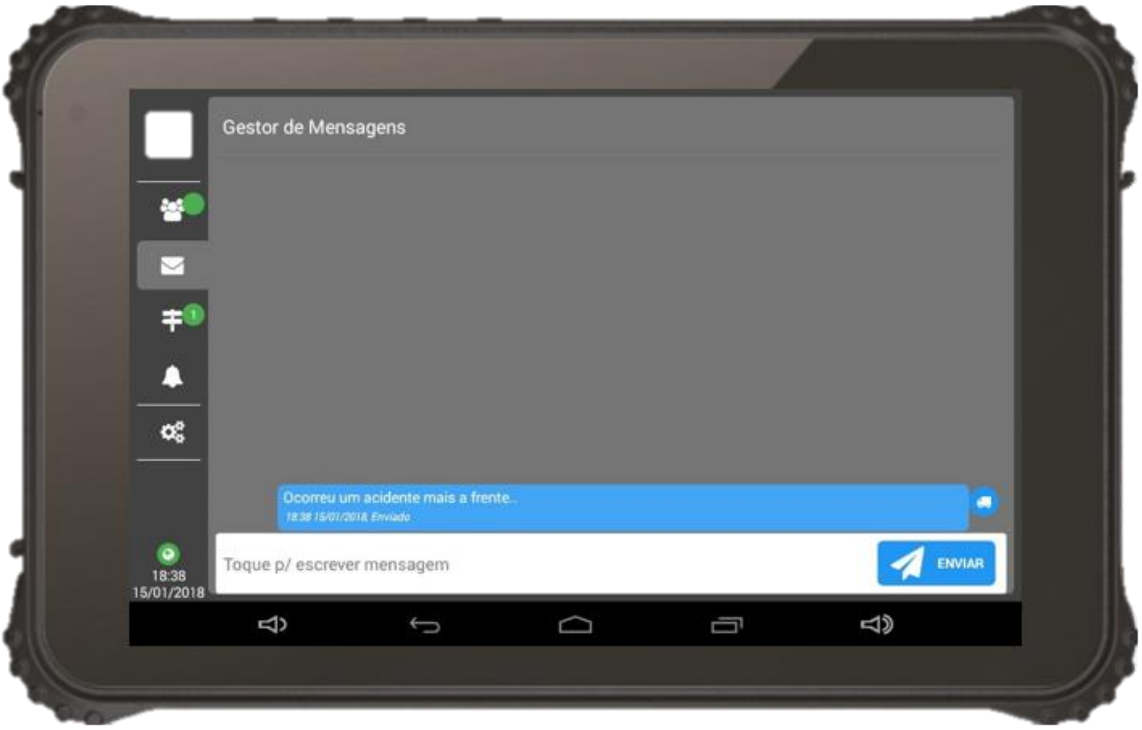


Figura 16 - Envio e receção de mensagem na consola gráfica do condutor;

Na **Figura 15** e **Figura 16** está representado a consola gráfica que os condutores usam para inserir e recolher informação sobre todo o negócio do mundo Tecmic. Com o exemplo da **Figura 16** com a caixa de mensagens.

Figura 17 - Terminal PDA;

No canto inferior direito da **Figura 14** está representado o equipamento onde os sistemas web são executados, seja por aplicações desktop ou por soluções web (*XTraN* e *iZiTraN web*, entre outros). Os pedidos destes clientes são enviados para o servidor *Web (Web Server)* que por sua vez comunica com o sistema central, *Application Server*. Seguem na **Figura 18** e na **Figura 19**, exemplos de funcionalidades e do design das soluções web existentes no negócio da empresa.

Figura 18 - Funcionalidades dos sistemas Web Tecmic;

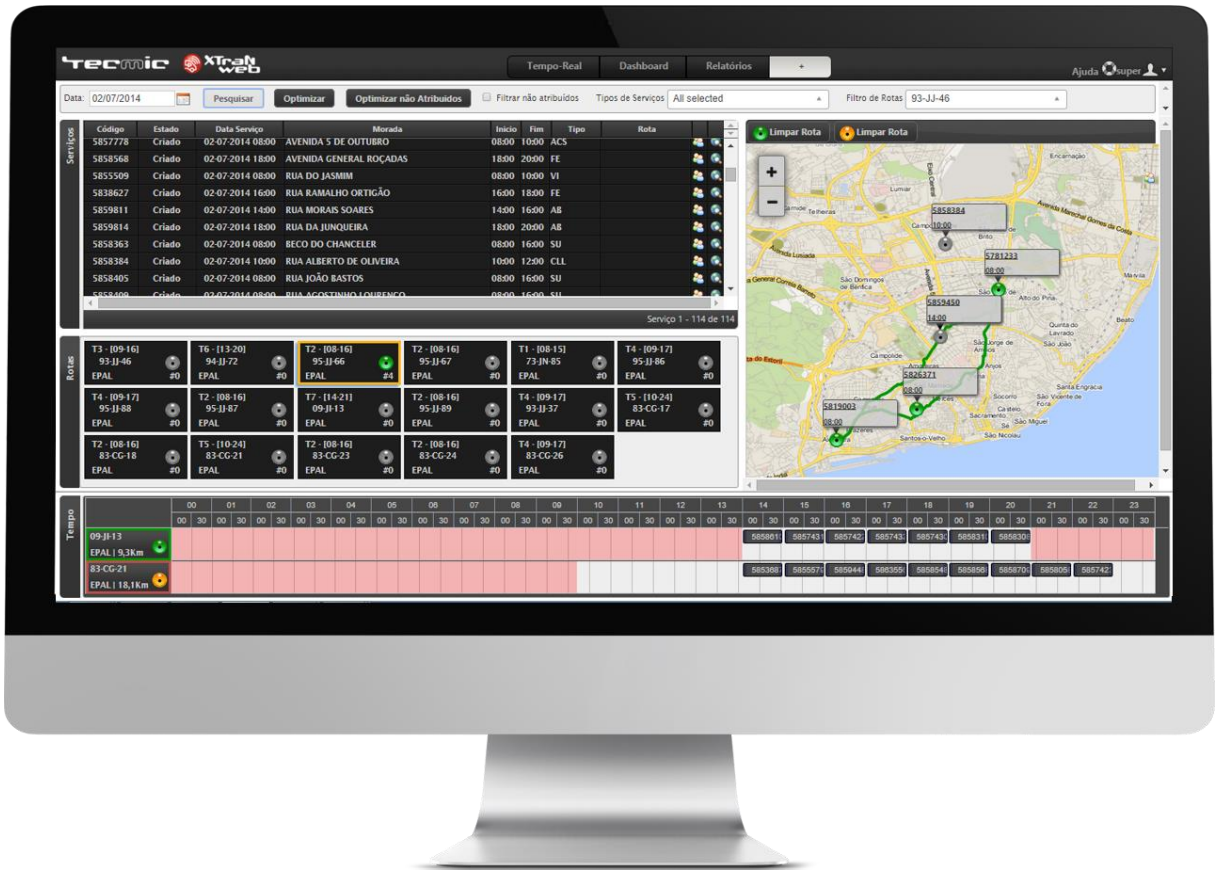


Figura 19 - Ecrã exemplo do xTraN Web;

Para além destes sistemas, a empresa Tecmic S.A. tem também uma forte componente de soluções Mobile, como a *Driver Mobile* e a *FleetMobile*, que estão representados na secção central inferior da **Figura 14** *(Mobile Clients)*. Com a exigência dos utilizadores houve a necessidade de migrar estes vários sistemas para plataformas Apple. Estas aplicações móveis *(Mobile Clients),* comunicam com o servidor móvel *(Mobile Server)*, criado especificamente para estes projetos, que interage com o *Application Server* existente. De notar que não se utilizou o *Web Server* existente (utilizado pelo *iZiTraN Web*) por este não estar ajustado às necessidades da aplicação em dispositivos móveis, devido ao forte impacto de tempo real que esta requer.

4.3.2 Servidor Mobile

Dando mais foco ao Servidor Mobile, pois este é o que comunica diretamente com a aplicação FleetMobile, a que motivou a redação deste documento. Esta aplicação contém uma aplicação servidora, que disponibiliza um conjunto de *Web services*, denominados *iZiTraN Mobile Services*.

O iZiTraN Mobile Services é composto por duas componentes, o *iZiTraN Mobile Service* e o *iZiTraN Mobile RealTime*. O primeiro é responsável por satisfazer os serviços pedidos pela aplicação cliente enquanto que o segundo para os serviços em tempo real, como o próprio nome indica. Segue a **Figura 20** com a representação destas duas componentes e as respetivas interações com as mesmas.

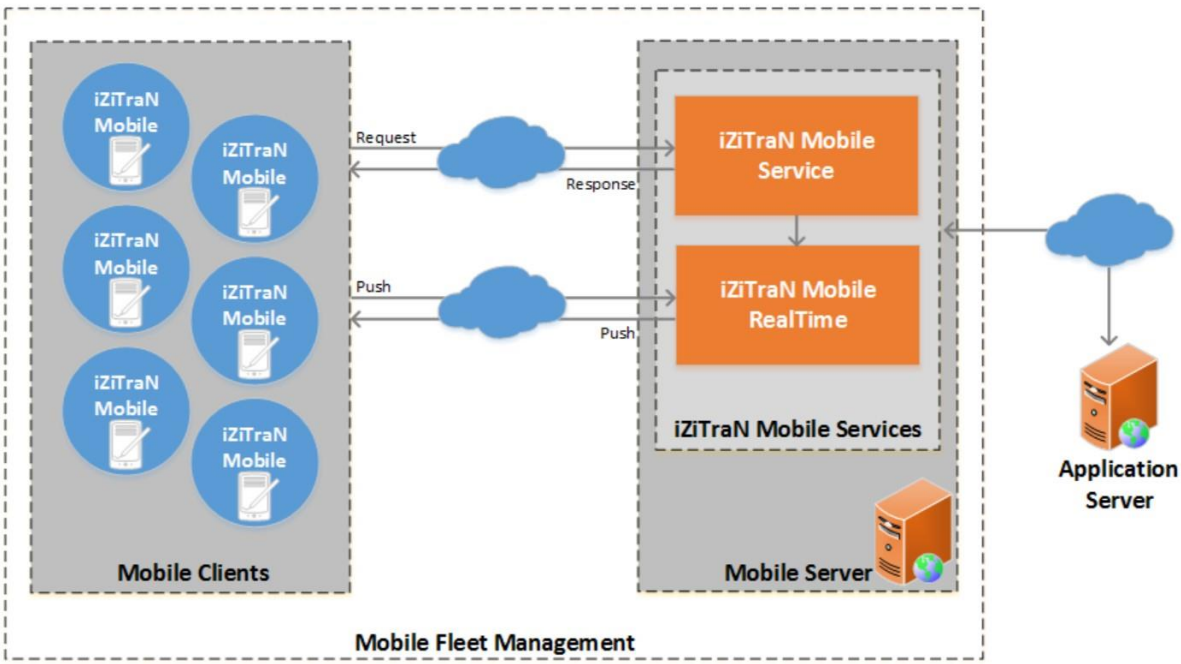


Figura 20 - Enquadramento arquitetual so iZiTraN Mobile Services [4];

Como pode ser visto na **Figura 20**, na secção *Mobile Clients* da mesma, os clientes encontram-se com o nome de "*iZiTraN Mobile*", foi dado este nome por forma a generalizar as aplicações moveis que a empresa disponibiliza no seu negocio, já previamente referidas na secção anterior. Resumidamente, o *iZiTraN Mobile Service* apresenta uma arquitetura *REST*. Em relação ao *iZiTraN Mobile RealTime* encontra-se integrado com a *framework SignalR* por esta tratar de comunicações bidirecionais, encontrando o transporte mais eficaz entre servidor e cliente, dando preferência que seja por *WebSockets*. [4][15]

Para a migração detalhada neste documento, da aplicação nativa para uma aplicação multiplataforma, não foi feita qualquer alteração ao servidor, como tal este manteve-se intacto. O detalhe do mesmo encontra-se na **referência [4] na secção 4.4**.

4.4 Modelo de Domínio

Esta secção será baseada na referência [4] que foi um documento escrito para o desenvolvimento da primeira versão desta aplicação, a versão Android, que foi agora migrada para uma aplicação multiplataforma. O diagrama geral do modelo de domínio é apresentado na **Figura 21**.

Como é possível ver no diagrama do modelo de domínio, existe um enorme número de classes e relações entre as mesmas, com isso é possível derivar que o processo de negócio da empresa Tecmic S.A. apresenta um elevado nível de complexidade. Como tal, o diagrama será explicado de forma fragmentada, dando a conhecer o modelo e as decisões associadas. O objetivo desta secção não é descrever o detalhe de todos os atributos das entidades, mas sim, descrever os atributos que compõem relações entre as diversas entidades do modelo de domínio.

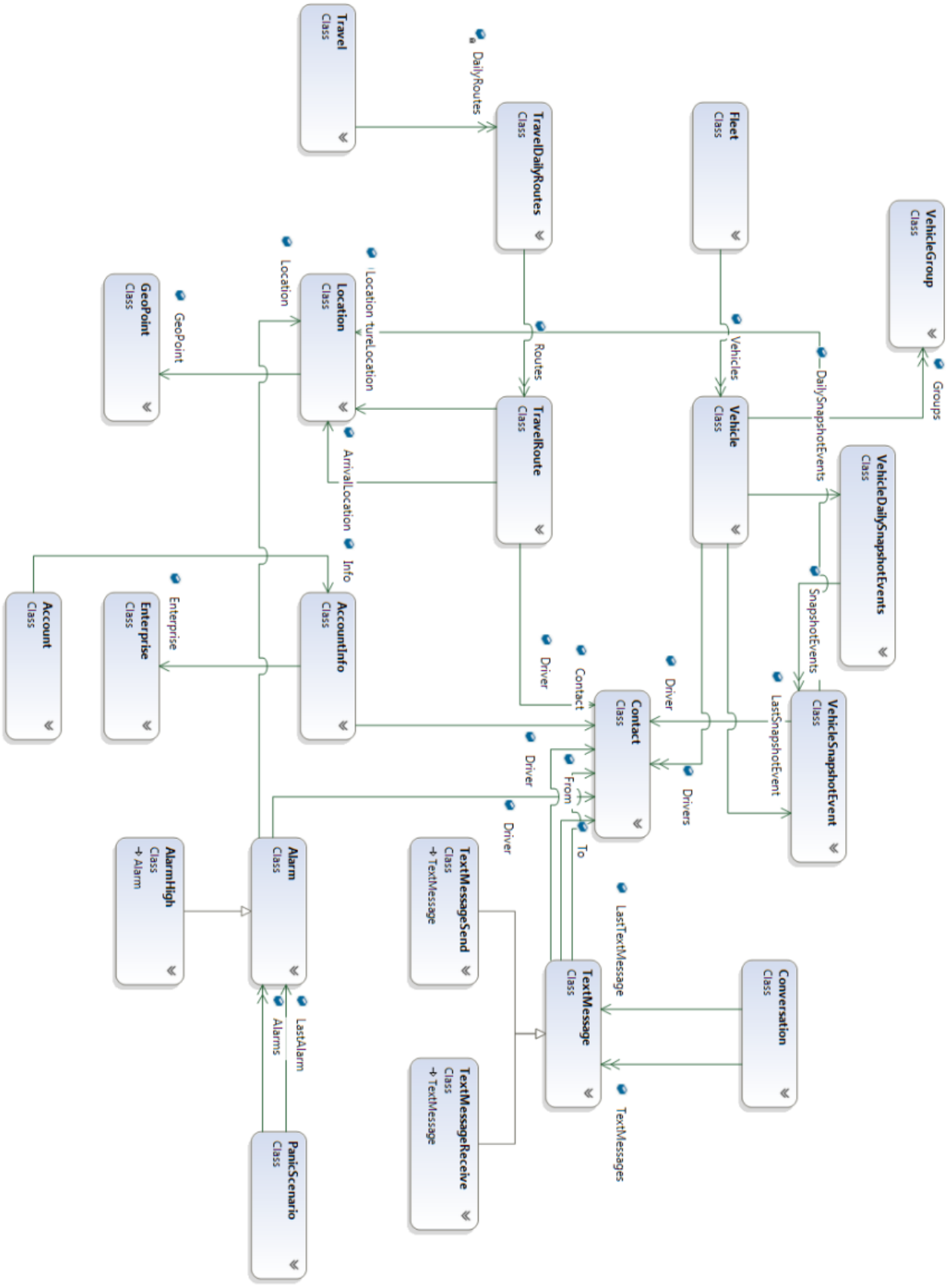


Figura 21 - Diagrama do modelo de dominio [4];

- ***Account***: esta entidade modela toda a informação da conta do utilizador no sistema. As dependências desta entidade são apresentadas na **Figura 22**;

Figura 22 - Entidade Account;

A gestão de contas é importante para permitir a identificação do utilizador no sistema e todo o conjunto de relações e entidades, permitiu assim ter uma plataforma multicliente. A conta tem várias informações vinculadas a si, conforme representa o atributo *Info* na conta.

- ***AccountInfo***: entidade que representa a informação abrangida pela conta, cujas dependências são apresentadas na **Figura 23**.

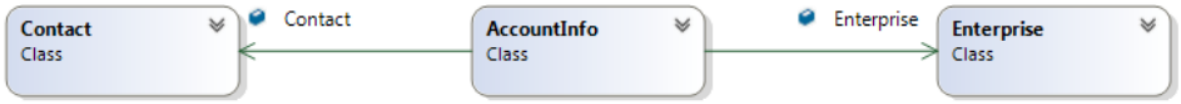


Figura 23 - Entidade AccountInfo;

A entidade ***AccountInfo*** não representa o conceito de conta, mas sim da informação associada à mesma. A informação da conta tem um contacto que identifica o utilizador, assim como a empresa a que esta pertence.

- ***Contact***: esta entidade representa o contacto a partir do qual o tipo de contacto (Operador de Frota, Condutor, Veículo, Terminal) pode ser identificado e contactado. Esta entidade não depende de nenhuma outra do modelo de domínio.
- ***Enterprise***: entidade que representa a empresa. Esta entidade não depende de nenhuma outra do modelo de domínio.

Como a lógica de negócio assenta sobre a gestão de frotas houve a necessidade da criação da entidade frota.

- ***Fleet***: esta entidade representa a frota de veículos. As dependências desta entidade são apresentadas na **Figura 24**.

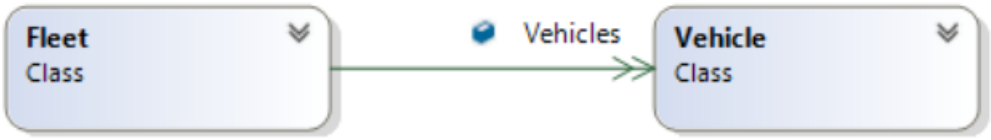


Figura 24 - Entidade Fleet;

A frota é um agrupamento de veículos, sendo estes um atributo fundamental em todo o modelo de domínio.

- ***Vehicle***: entidade que modela toda a informação referente ao veículo, cujas dependências são apresentadas na **Figura 25**.

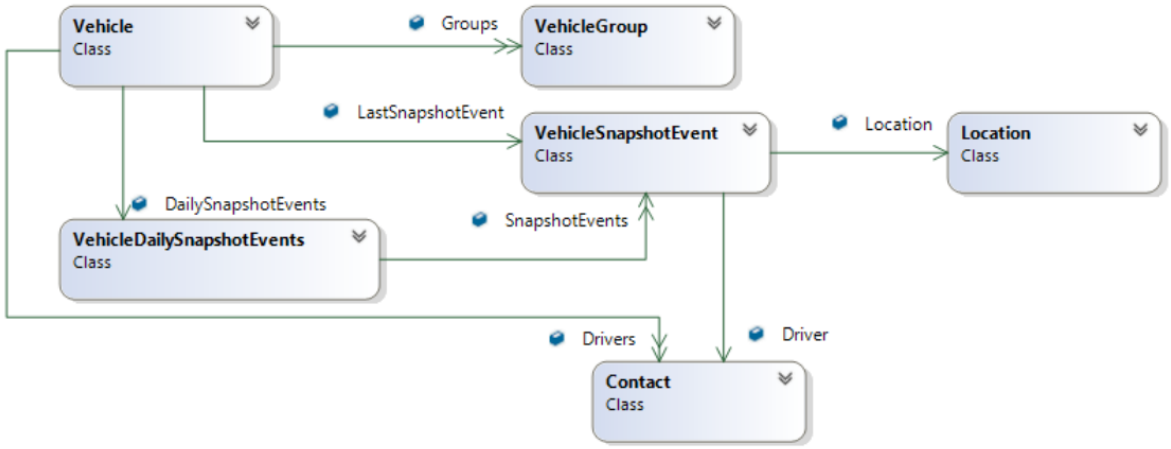


Figura 25 - Entidade Vehicle, VehicleGroup, VehicleDailySnapshotEvents;

Um veículo pode pertencer a vários grupos que são criados para facilitar a sua correta organização por zonas. O veículo tem estatísticas diárias que são totalizadas através dos eventos ocorridos, pelo mesmo, ao longo do dia.

- ***VehicleGroup:*** entidade que representa o grupo de veículos. Esta entidade não depende de nenhuma outra do modelo de domínio.

- ***VehicleDailySnapshotEvents:*** entidade que contem todas as estatísticas diárias do veículo. Tem o atributo ***SnapshotEvents***, que cria uma dependência com a entidade ***VehicleSnapshotEvents***, com o intuito de criar as estatísticas diárias.

- ***VehicleSnapshotEvent***: esta entidade representa os indicadores instantâneos do veículo, como estado, velocidade instantânea, percentagem de combustível, condutor ativo, entre outros. Esta entidade tem uma relação para ***Contact*** contendo o condutor e para ***Location***.

- ***Location***: entidade que representa a localização de um veículo, podendo ser de um dado instante, pontos de uma viagem ou de um alarme ocorrido. As dependências desta entidade geográfica são apresentadas na **Figura 26**.

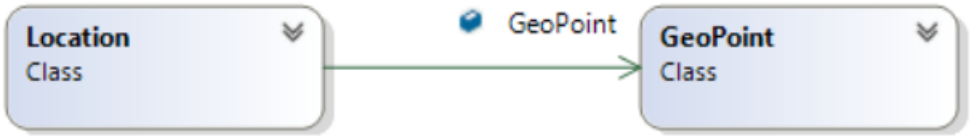


Figura 26 - Entidade Location;

Uma localização tem posição descritiva e coordenadas geográficas para representar a mesma.

- ***GeoPoint***: esta entidade representa o ponto de localização na forma de coordenadas geográficas através da latitude e longitude. Esta entidade não depende de nenhuma outra do modelo de domínio.

Como os veículos efetuam ao longo do dia várias viagens houve a necessidade da criação da entidade viagem.

- ***Travel***: entidade que representa a viagem efetuada por um veículo, cujas dependências são apresentadas na **Figura 27**.

Figura 27 - Entidade Travel;

Os veículos efetuam viagens todos os dias, fazendo com que a entidade viagem seja um agrupamento de viagens diárias.

- ***TravelDailyRoutes***: esta entidade representa a viagem diária que um veículo faz. As dependências desta entidade são apresentadas na **Figura 28**.

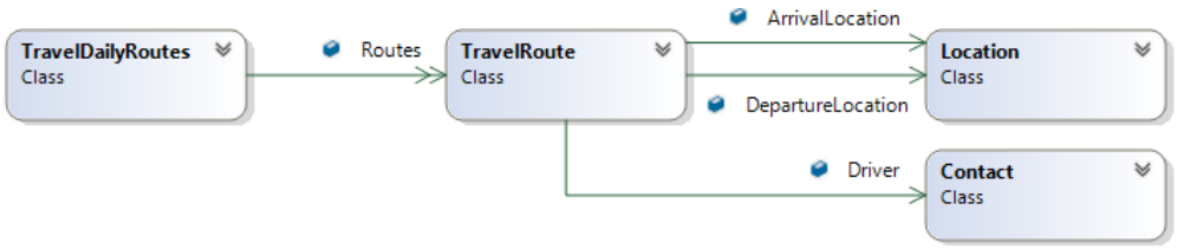


Figura 28 - Entidade TravelDailyRoutes e TravelRoute;

As viagens diárias são um agrupamento dos percursos (trajetos efetuados com a ignição ligada) efetuados pelos veículos naquele dia.

- ***TravelRoute***: entidade que representa o percurso efetuado. Esta entidade depende da entidade ***Location*** para conhecer o local de chegada e destino, bem como da ***Contact*** para conhecer o condutor que efetuou aquele percurso.

Como os veículos podem ao longo do dia desencadear vários alarmes houve a necessidade da criação da entidade cenário de pânico.

- ***PanicScenario***: esta entidade representa o cenário de pânico, sendo utilizada para agrupar alarmes, cujas dependências são apresentadas na **Figura 29**.

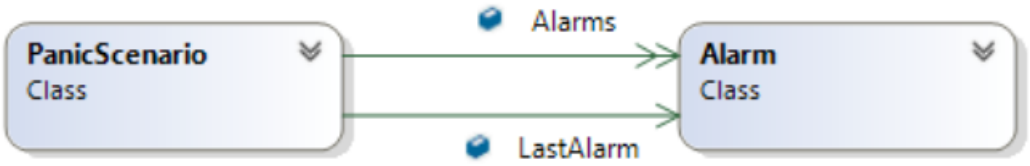


Figura 29 - Entidade PanicScenario;

O cenário de pânico é um agrupamento dos alarmes desencadeados pelos veículos. Esta entidade tem ainda o último alarme desencadeado.

- ***Alarm***: entidade que representa o alarme desencadeado por um veículo. As dependências desta entidade são apresentadas na **Figura 30**.

- ***AlarmHigh***: esta entidade representa o alarme com alta prioridade. Esta entidade herda de **Alarm** e tem como particularidade o facto de necessitar de ser tratado.

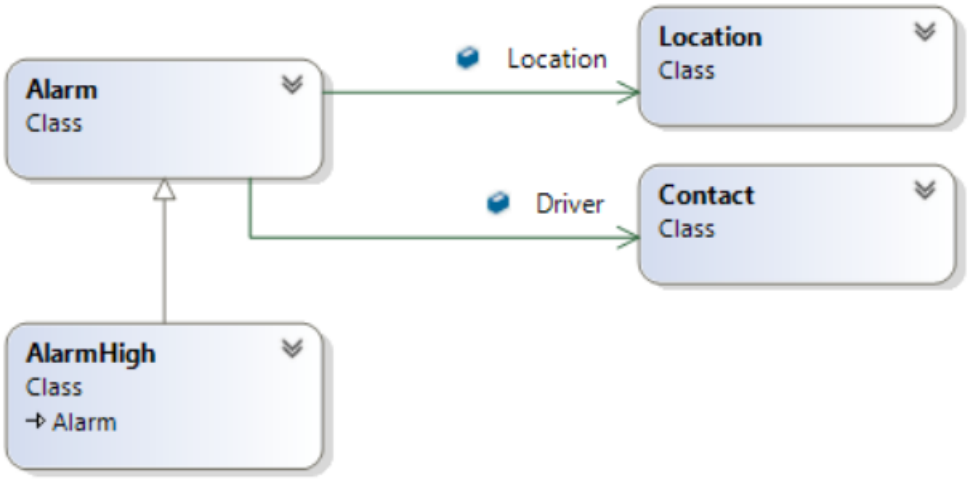


Figura 30 - Entidade Alarm e AlarmHigh;

Como os veículos e condutores são contactáveis, sendo que por este motivo podem enviar e receber mensagens de texto, houve a necessidade da criação da entidade conversa.

- ***Conversation***: entidade que representa a conversa trocada através de mensagens, cujas dependências são apresentadas na **Figura 31**.

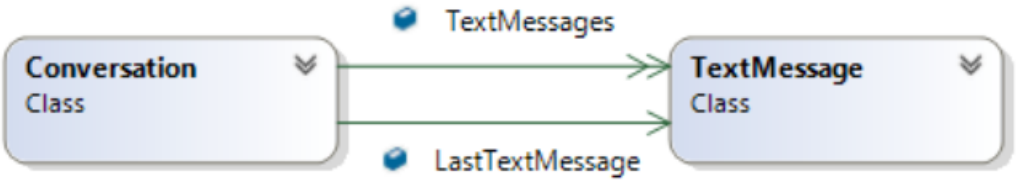


Figura 31 - Entidade Conversation;

A conversa é um agrupamento das mensagens enviadas e recebidas. Esta entidade tem ainda a última mensagem da conversa.

- ***TextMessage***: esta entidade representa a mensagem de texto. As dependências desta entidade são apresentadas na **Figura 32**.

- ***TextMessageSend***: entidade que representa a mensagem enviada. Esta entidade herda de ***TextMessage*** e tem como particularidade o facto ter os estados: criada, guardada, enviada, entregue, vista e de erro, bem como as datas de cada estado.

- ***TextMessageReceive***: esta entidade representa a mensagem recebida. Esta entidade herda e ***TextMessage*** e tem como particularidade o facto ter os estados: recebida, não lida e lida, bem como as datas de cada estado.

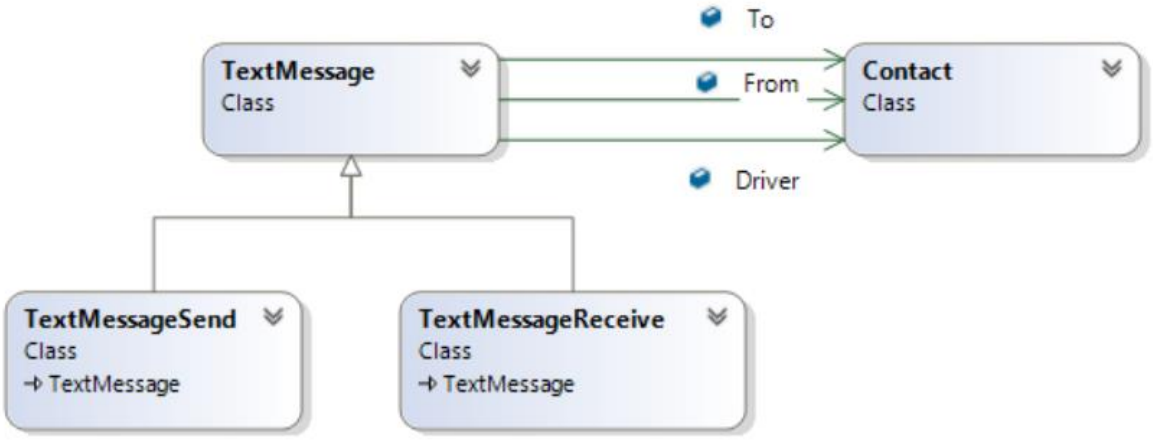


Figura 32 - Entidade TextMessage, TextMessageSend e TextMessageReceive;

Sendo o modelo estruturado de forma a ser o mais estável, flexível, extensível, manutenível e escalável possível, possibilitou que as várias aplicações e sistemas partilhassem o mesmo modelo de domínio. Como referido no início desta secção, esta secção foi puramente baseada na referência [4], que foi um documento elaborado para o desenvolvimento da primeira versão nativa desta aplicação, a versão Android.

4.5 Aplicação Móvel

A aplicação FleetMobile, como já referido anteriormente, permite ao operador de frota, com segurança, obter acesso ao estado dos veículos e realizar ações sobre os mesmos. Este pode visualizar em lista e em mapa o estado dos veículos (desligado, parado, em trânsito, em movimento), ter acesso a vários indicadores como localização, velocidade, quilómetros percorridos, consumos de combustível, entre outros. É também possível visualizar o histórico de viagens do veículo, o envio e receção de mensagens e os alarmes que podem ser desencadeados pelos veículos.

4.5.1 Arquitetura das várias camadas

Nesta secção será comparada a arquitetura da primeira versão da aplicação com a arquitetura depois da integração com o *Xamarin* e o *Mvvm-Cross*. Na **Figura 33**, encontra-se representada a arquitetura da versão nativa Android da aplicação que foi migrada.

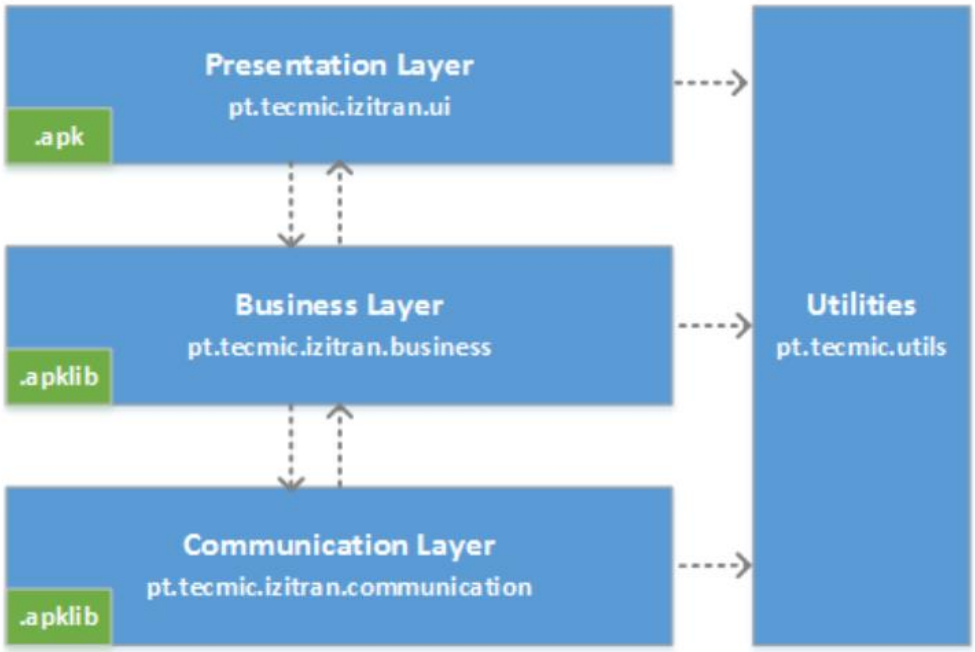


Figura 33 - Arquitetura da aplicação FleetMobile nativa Android;[4]

Como já foi referido, a **Figura 33** é representativa da arquitetura da versão Android nativa da aplicação que foi migrada, FleetMobile. Esta arquitetura é composta por três camadas principais. A camada de apresentação/vista corresponde à “Presentation Layer”. A camada de negócio, “Business Layer”, contem todas as regras de negócio que determinam quais os dados criados, transformados ou persistidos que seguidamente poderão ir, dependendo da regra de negócio, para a camada de apresentação/vista ou para a camada de comunicação, “Communication Layer”.

A versão multiplataforma, por sua vez, foi desenvolvida utilizando a *framework* Xamarin em simultâneo com a framework *Mvvm-Cross*, mais numa lógica de modulação de dados e passagem

dos mesmos entre camadas. Assim sendo a aplicação FleetMobile multiplataforma ficou com a seguinte arquitetura, ver a **Figura 34**.

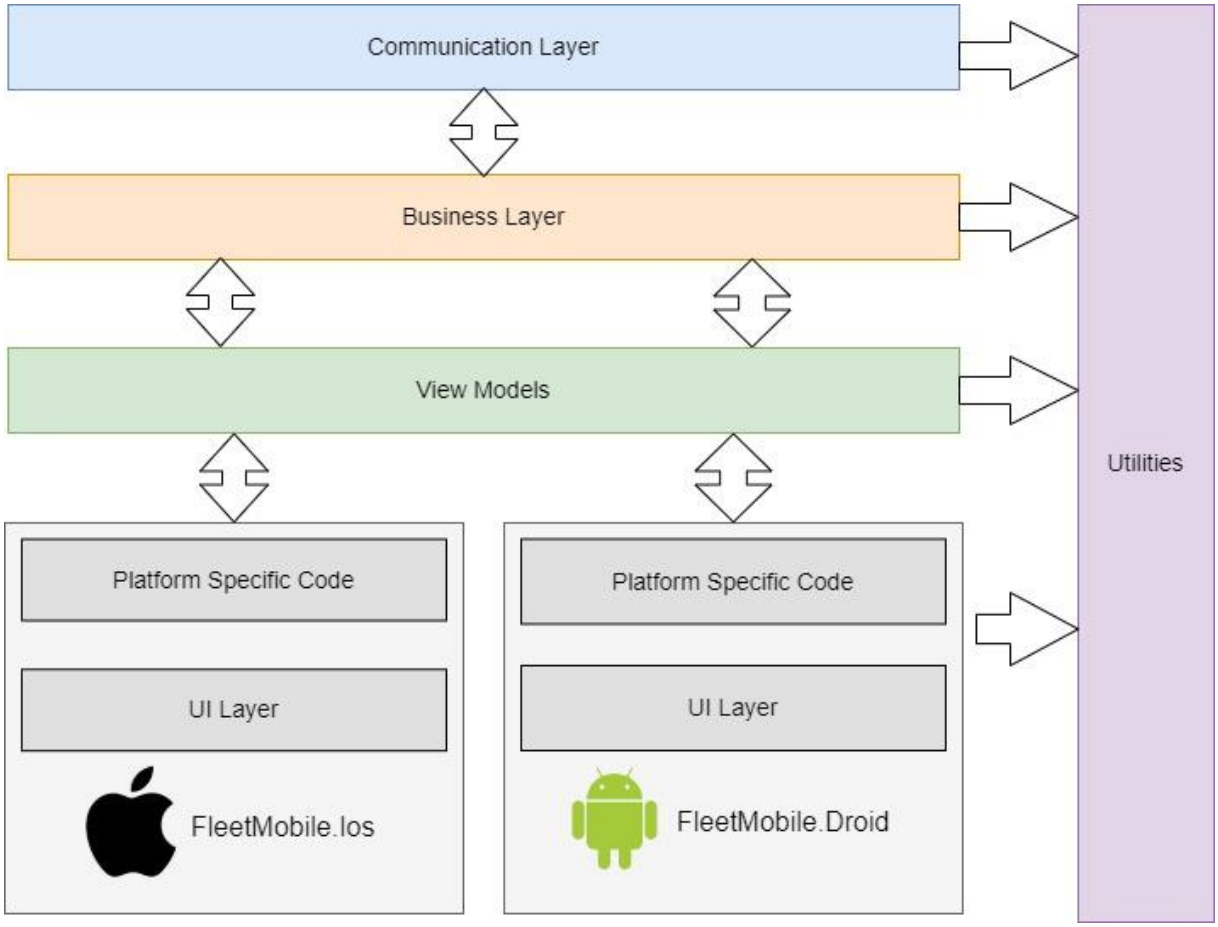


Figura 34 - Arquitetura da aplicação FleetMobile usando Xamarin e Mvvm-Cross*;*

Tanto na **secção 2.2.2** como na **secção 2.4.2**, está detalhado estas duas *frameworks*. as mesmas combinadas deram origem à arquitetura da **Figura 34**. Que será explicado de seguida a origem da mesma.

O *Xamarim*, **secção 2.2.2**, apresenta uma arquitetura onde todo o código é aproximadamente 96% reutilizável e comum a ambas as plataformas, Android e iOS, onde as camadas de negócio, comunicações e acesso a dados é a mesma, como pode está detalhado na **Figura 34** com as camadas "*Business Layer*" e "*Communication Layer*". Enquanto que as camadas não comuns, são as especificas de cada plataforma, "*FleetMobile.Droid" e "FleetMobile.Ios*", que são consideradas as camadas de vista.

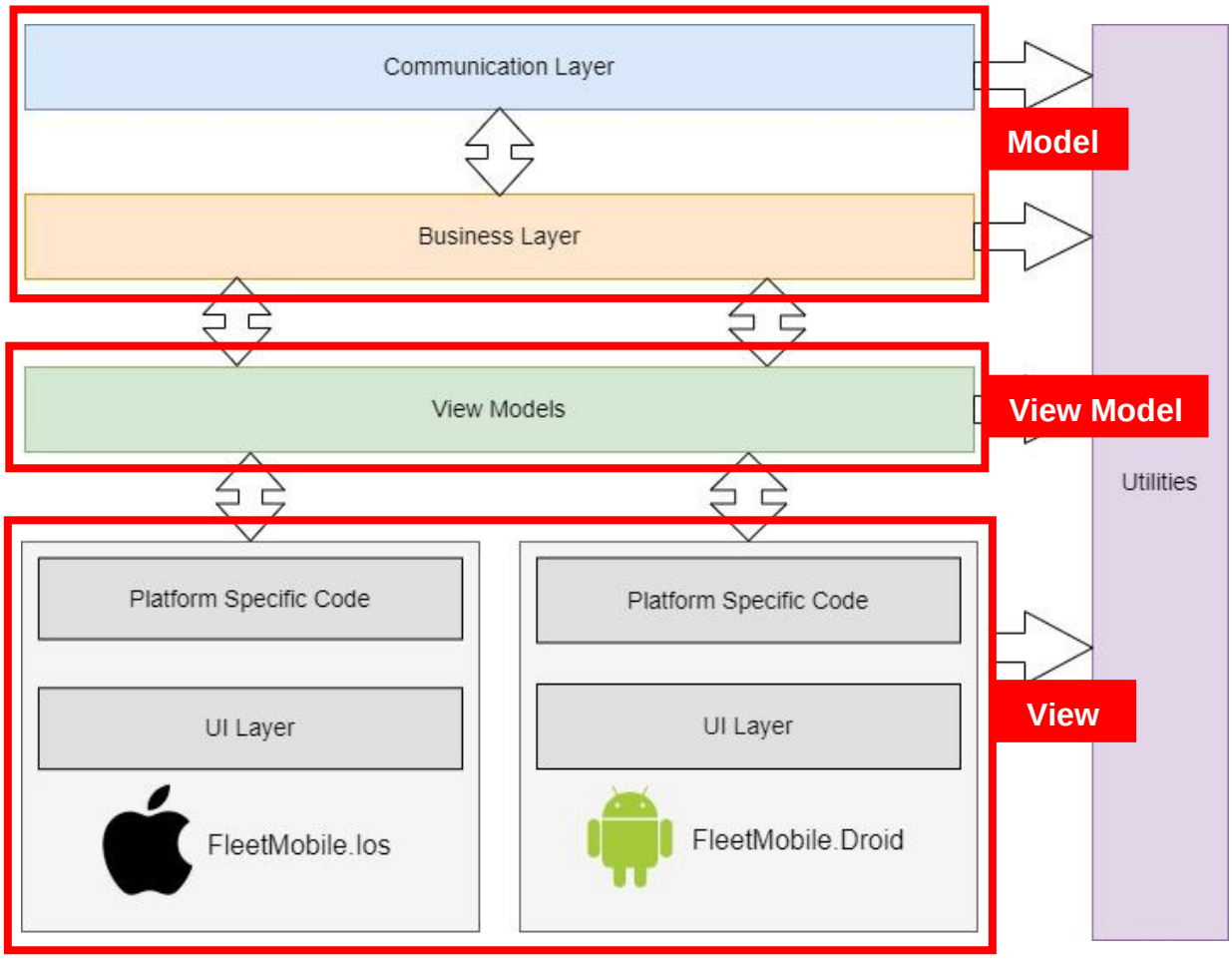


Figura 35 - Arquitetura das camadas da FleetMobile esquematizada com a framework Mvvm-Cross*;*

Com a necessidade de diminuir a complexidade e o esforço de manutenção das aplicações das várias plataformas, integrou-se o *Mvvm-Cross*, **secção 2.4.2**, que é uma *framework* que, resumidamente, usa o padrão de desenho *MVVM* que se traduz em *Model – View – ViewModel*. A componente ***Model*** deste padrão de desenho foi traduzida e utlizada na camada "*Business Layer*" e "*Communication Layer*". A componente ***View*** nas camadas de vista especificas de cada plataforma, "*FleetMobile.Droid" e "FleetMobile.Ios*". Enquanto que a componente ***ViewModel*** obrigou a criar uma nova camada, a "*View Models*", sendo esta também comum a todas as plataformas. Desta forma, existe uma separação clara entre as implementações de interface e de negócio, além de permitir o desenvolvimento de testes unitários de forma simples.

Todas as camadas enunciadas têm a sua própria *package*, ou seja, cada uma tem o seu próprio conjunto de ficheiros de código que compõem cada camada, servindo assim os propósitos da elaboração deste projeto de conter o código reutilizável, proporcionar agilidade em futuras ações de manutenção e atualização de forma uniforme para os dois sistemas alvo de interesse. A camada de negócio, "*Business Layer*", contem todas as regras de negócio que determinam quais os dados criados, transformados ou persistidos que seguidamente poderão ir (dependendo da regra de negócio) para a camada de apresentação/vista, *"FleetMobile.Droid" e "FleetMobile.Ios",* ou para a

camada de comunicação, *"Communication Layer".* Desta forma, a camada de negócio funciona como um interpretador de dados. Por outro lado, a camada de comunicação incluí as classes que fazem parte da comunicação entre a aplicação e os seus *Web services*, como com a *framework* de *SignalR* responsável pelas comunicações das várias funcionalidades em tempo real.

A camada "*Utilities*", é representativa de todas as bibliotecas externas que auxiliam o desenvolvimento da aplicação, desde conversores de tipos de dados, a bibliotecas de cores para que sejam uniformes a todas as plataformas, a conversores dos vários objetos e tipos de dados do modelo de domínio, entre outros.

4.5.2 Guidelines de Interfaces

Para o desenvolvimento de interfaces de qualquer aplicação ou sistema é essencial seguir as várias diretrizes de usabilidade e desenho das várias plataformas. Para uma aplicação mobile multiplataforma, é necessário então validar as diferentes *guidelines*, neste caso concreto as *guidelines* tanto de Android como de iOS.

Para o desenvolvimento da aplicação Android nativa, foram seguidas as *guidelines* de Android que estão detalhadas na **secção 4.5.1.2** da referência [4]. Como tal, nesta nova aplicação multiplataforma, reutilizou-se as interfaces que já existiam da aplicação nativa Android. Por outro lado, para a versão iOS, foi necessário estudar aprofundadamente as *guidelines iOS*, até porque não havia qualquer conhecimento ou experiência na equipa de desenvolvimento da Tecmic S.A., pode-se portanto dizer que este projeto foi o projeto piloto para o desenvolvimento iOS.

Numa forma geral existem alguns princípios base que ajudam a criar impacto e a vincar a identidade de uma aplicação seja ela Android ou iOS, são eles [19]:

- ***Aesthetic Integrity –*** A integridade estética representa o quão bem a aparência e o comportamento de uma aplicação se integram à sua função. Por exemplo, uma aplicação com um teor profissional, que ajuda os utilizadores a executar uma tarefa séria, para os cativar e mantê-los concentrados é por norma usual usar gráficos sutis e discretos, controlos padrão e comportamentos previsíveis. Por outro lado, uma aplicação imersiva, como um jogo, por norma oferece uma interface cativante que promete entusiasmo e emoção, além de incentivar a descoberta.

- ***Consistency –*** Uma aplicação consistente implementa padrões e paradigmas familiares usando elementos de interface fornecidos pelo sistema, ícones conhecidos, estilos de texto padrão e terminologia uniforme. Sendo que estes padrões devem ser comuns em toda a aplicações, seja tipos de letra, seja cores ou mesmo ícones.

- ***Feedback –*** Outro princípio relevante é o feedback. As aplicações devem reconhecer ações e mostrar resultados por forma a manter os utilizadores informados sobre as suas ações. As

aplicações iOS fornecem feedback percetível em resposta a cada ação do utilizador. Por exemplo, as componentes interativas no momento da interação devem-se destacar, os indicadores de progresso mostram o status das operações de longa execução, e algumas animações e sons ajudam a esclarecer os resultados das ações. Quando é colocado algum campo de entrada com informação inconsistente, a aplicação deve dar feedback sobre o erro.

- ***Metaphors –*** Uma das melhoras formas de tornar uma interface intuitiva e com um grau de usabilidade bom é usando metáforas. Os utilizadores aprendem mais rapidamente quando os objetos de uma aplicação são metáforas de experiências familiares. As metáforas funcionam bem no iOS porque as pessoas interagem fisicamente com o ecrã. Por exemplo, para identificar um veículo, usar um icon de veículo. Uma mensagem usar uma carta aberta ou fechada consoante se a mensagem já foi lida ou não.

- ***User Control*** – Nos sistemas iOS, os utilizadores – não as aplicações – estão no controlo. Uma aplicação pode sugerir uma ação ou alertar sobre consequências perigosas, mas geralmente é um erro uma aplicação assumir a tomada de decisões. Os utilizadores têm que ter toda a liberdade para operar sobre as aplicações. As melhores aplicações encontram o equilíbrio correto entre habilitar os utilizadores e evitar resultados indesejados. Uma aplicação deve fazer com que os utilizadores sintam que estão no controlo, mantendo os elementos interativos familiares e previsíveis, confirmando ações destrutivas e facilitando o cancelamento de operações, mesmo quando já estão em andamento. Resumindo, os utilizadores têm de ter toda a liberdade sobre as aplicações.

Existem três princípios que a Apple afirma que diferencia as plataformas iOS de todas as outras plataformas [19], são estes:

- ***Clarity –*** O primeiro princípio denomina-se clareza e diz que em todo o sistema, o texto tem de ser legível em todos os tamanhos, os ícones têm que ser precisos e lúcidos, os retoques de estilização têm que ser sutis e apropriados, como os sombreamentos e as formas das várias componentes. Espaço negativo, cor, fontes, gráficos e elementos de interface têm de destacar sutilmente conteúdos importantes e transmitir interatividade.

- ***Deference -*** O movimento fluido e uma interface nítida e simples ajudam os utilizadores a entender e interagir com o conteúdo sem nunca entrar em conflito com o mesmo. As várias interfaces normalmente preenchem o ecrã inteiro, em simultâneo, a translucidez e o desfoque geralmente acentuam mais. O uso mínimo de molduras, gradientes e sombras mantém a interface leve e arejada, garantindo ao mesmo tempo o valor do conteúdo.

- ***Depth -*** Camadas visuais distintas e movimentos realistas transmitem hierarquia e facilitam o entendimento. O toque e a capacidade de descoberta aumentam o prazer e permitem o

acesso à funcionalidade e ao conteúdo adicional sem perder o contexto. As transições fornecem uma sensação de profundidade à medida que o utilizador navega pela interface.

4.5.3 Interfaces Gráficas

Como foi referido na secção anterior, todas as interfaces Android já se encontravam criadas e como tal foram reutilizadas. Em relação ao desenho das interfaces para a plataforma iOS, utilizou-se a ferramenta ***Interface Builder*** do **XCode** que, como foi detalhado na **secção 2.2.1**, resumidamente funciona com uma logica de **drag-and-drop** de componentes. Esta ferramenta utiliza a *framework UIKit* nativa de iOS. Esta *framework* permite que as aplicações obtenham uma interface consistente no total do sistema, oferecendo também um grande nível de customização. Os elementos *UIKit* são flexíveis e familiares. São adaptáveis, permitindo criar uma única aplicação preparada para qualquer dispositivo e ecrã iOS, para além que são atualizados automaticamente quando o sistema introduz alterações na aparência.

Na *framework UIKit* as várias componentes encontram-se categorizadas em três tipos [19]:

- ***Bars*** – Componentes que informam os utilizadores sobre a sua localização na aplicação, forneça navegação e podem conter botões ou outros elementos para iniciar ações e comunicar informações:

- ***Views*** – Componentes que funcionam como contentores do conteúdo principal do que os utilizadores veem na sua aplicação, como texto, gráficos, animações e elementos interativos. As *views* permitem comportamentos de scroll, inserção, exclusão e organização;

- ***Controls*** – Componentes que permitem despoletar ações e transmitir informações. Por exemplo botões, *switches*, campos de texto e indicadores de progresso são exemplos dessas componentes;

Posteriormente as interfaces desenvolvidas no *XCode* foram importadas no projeto do *Visual Studio*, conseguindo assim ter interfaces coerentes e que seguissem os estilos e as *guidelines* da Apple.

Antes da sua implementação foram desenhados alguns protótipos não funcionais para estudar possíveis alterações entre as interfaces já existentes Android e as novas interfaces iOS, pois as *guidelines* de iOS e de Android são algo diferentes. Após a validação dos mesmos procedeu-se á implementação das interfaces finais. De seguida são apresentadas algumas figuras das interfaces iOS contrastando com as Android (**Figura 36, Figura 37, Figura 38, Figura 39, Figura 40** e **Figura 41**).

Todas as capturas de ecrã das interfaces desenhadas encontram-se no **Anexo 4**.

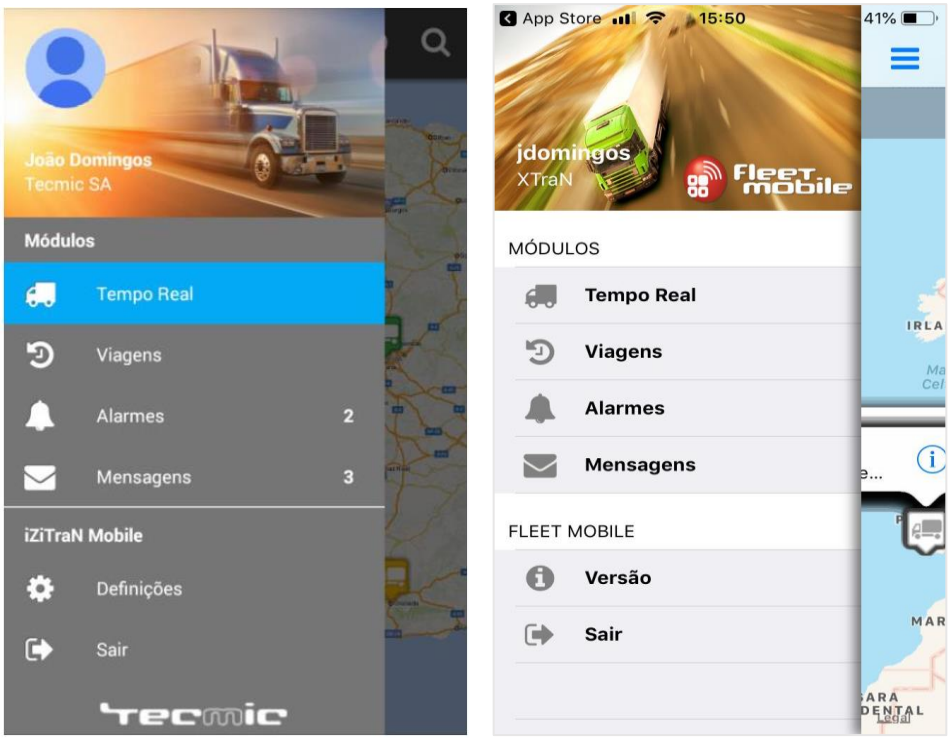


Figura 36 - Menu lateral de navegação versão Android (esquerda) iOS (direta);

A **Figura 36** apresenta à esquerda a versão existente Android e à direita a versão criada iOS, este ecrã apresenta o menu principal da aplicação, que permite navegar entre todos os módulos a que o utilizador tem acesso. No caso concreto desta figura, o utilizador tem acesso ao modulo “Tempo Real” – modulo onde é possível ver toda a sua frota num mapa. Modulo “Viagens” – modulo que permite ver todos os percursos e viagens dos veículos da sua frota. Modulo “Alarmes” – modulo que permite ver todos os alarmes emitidos pelos veículos da sua frota. Modulo “Mensagens – modulo que permite ver todos as conversas entre o utilizador e os veículos da sua frota. Neste menu também é possível aceder às preferências do utilizador e terminar a sessão.

Este menu pode ser acedido através de um botão, ☰, no canto superior esquerdo ou por deslizar o ecrã da borda esquerda para o centro. Estas formas de acesso ao menu encontram-se disponíveis em todos os ecrãs da aplicação, seguindo assim o princípio ***User Control*** detalhado na secção anterior deste documento, com o objetivo de dar toda a liberdade ao utilizador de aceder ao menu em qualquer ecrã da aplicação. Também segue o princípio ***Metaphors*** para representação do botão, foi escolhido o icon ☰ em vez do texto “Menu”.

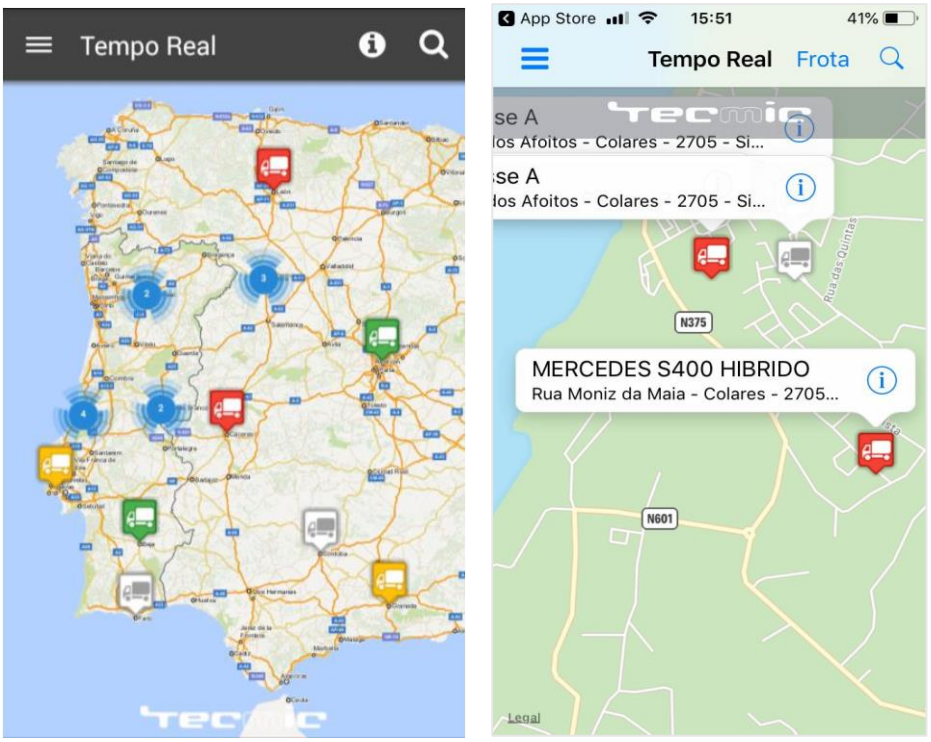


Figura 37 - Mapa com toda a frota, versão Android (esquerda) iOS (direta);

A **Figura 37** apresenta à esquerda a versão existente Android e à direita a versão criada iOS, este ecrã representa a frota toda do cliente no mapa, mostrando o estado do mesmo (cinza – estacionado, vermelho – parado, amarelo – em trânsito, verde – em movimento). A versão iOS apresenta também um pop-up com a informação do local em que este se encontra e o nome do mesmo, tendo a opção de clicar no botão ❗ navegando para um ecrã onde se encontra todos os indicadores atuais do veículo, desde velocidade instantânea, consumos, quilómetros, entre outros indicadores que serão descritos futuramente. Foi também utilizado *clusters* para agrupar vários veículos por forma a tornar mais simples a representação do mesmo, como pode ser visto neste caso na versão Android.

A versão Android utiliza mapas criados pela empresa Tecmic, utilizando o software *GeoServer*. Enquanto que na versão iOS utilizou-se os mapas nativos da *framework UIKit*, para termos acesso a todos os controlos dos mapas nativos iOS, seguindo assim o princípio de ***Consistency*** detalhado na secção anterior deste documento, obtendo assim consistência entre todas as aplicações iOS que utilizem mapas. Também segue o princípio ***Metaphors*** para representação, tanto dos veículos com 🚚 como das cores para diferenciar os vários estados do veículo.

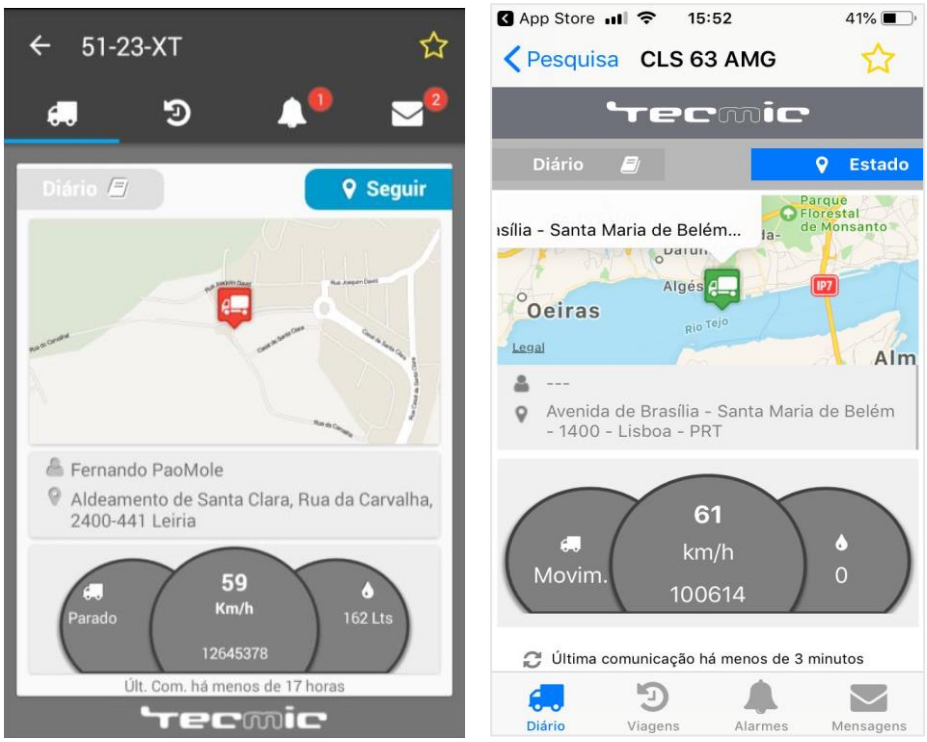


Figura 38 - Detalhe do veículo em tempo real, versão Android (esquerda) iOS (direta);

A **Figura 38** apresenta à esquerda a versão existente Android e à direita a versão criada iOS, este ecrã apresenta a opção "Diário" que corresponde aos indicadores diários do veículo, como quilómetros percorridos, velocidade média, consumo diário, entre outros. Já a opção "Seguir" (opção selecionada na **Figura 38**) corresponde a indicadores instantâneos. Este ecrã apresenta os indicadores atuais do veículo.

- **Localização atual** – Pode ser visto tanto no mapa como na informação ao lado do ícone 📍;
- **Condutor** – O nome do condutor encontra-se ao lado do ícone 👤;
- Painel do veículo
    - **Painel esquerdo** – Informação sobre o estado do veículo;
    - **Painel central** – Velocidade instantânea (em cima) e quilómetros (em baixo);
    - **Painel direita** – Informação do consumo;

De notar que todos estes indicadores são atualizados ao longo do tempo, sempre que haja alterações no mesmo utilizando a ***framework SignalR***.

Comparando as duas versões, verificou-se um cuidado acentuado principalmente no alinhamento da informação do painel do veículo. Como também na transposição do elemento de navegação por separadores do topo do ecrã para baixo, como é mais usual nas *guidelines* de iOS.

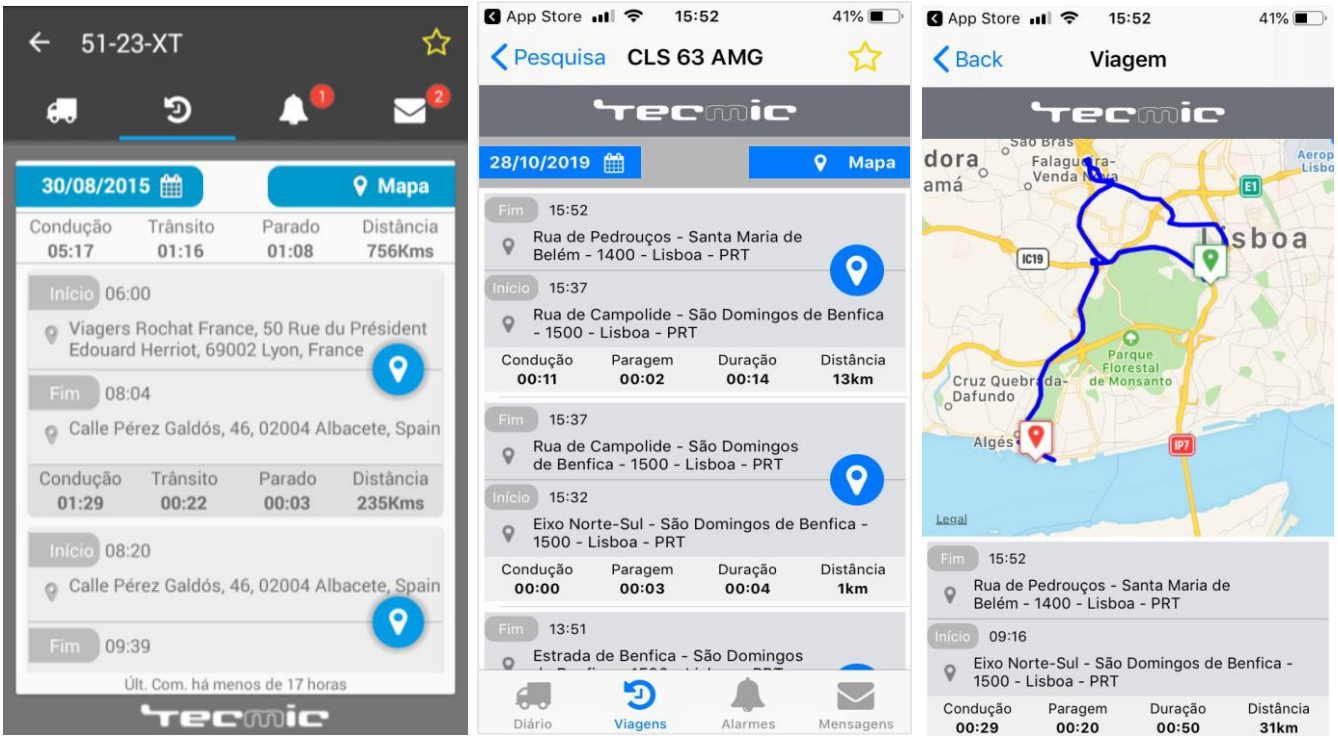


Figura 39 - Ecrã viagens de um veículo, versão Android (esquerda) seguida da versão iOS e à direta a representação da viagem em mapa na versão iOS;

A **Figura 39** apresenta à esquerda a versão existente Android e à direita a versão criada iOS. Este ecrã permite visualizar as viagens que o veículo selecionado efetuou ao longo do dia atual, como também a apresentação de viagens decorridas numa data específica, mostrando o seu trajeto em mapa e os respetivos indicadores. Estes indicadores correspondem à data, localização e tempos decorridos de início e fim de viagem que tanto podem ser de tempo em que o veículo esteve parado, em trânsito ou em movimento. [4]

De notar que em todos os ecrãs específicos de um veículo existe uma ☆ que permite adicionar o veículo em questão aos favoritos, facilitando assim a pesquisa do mesmo. Aquando o clique, o ícone passa a ⭐.

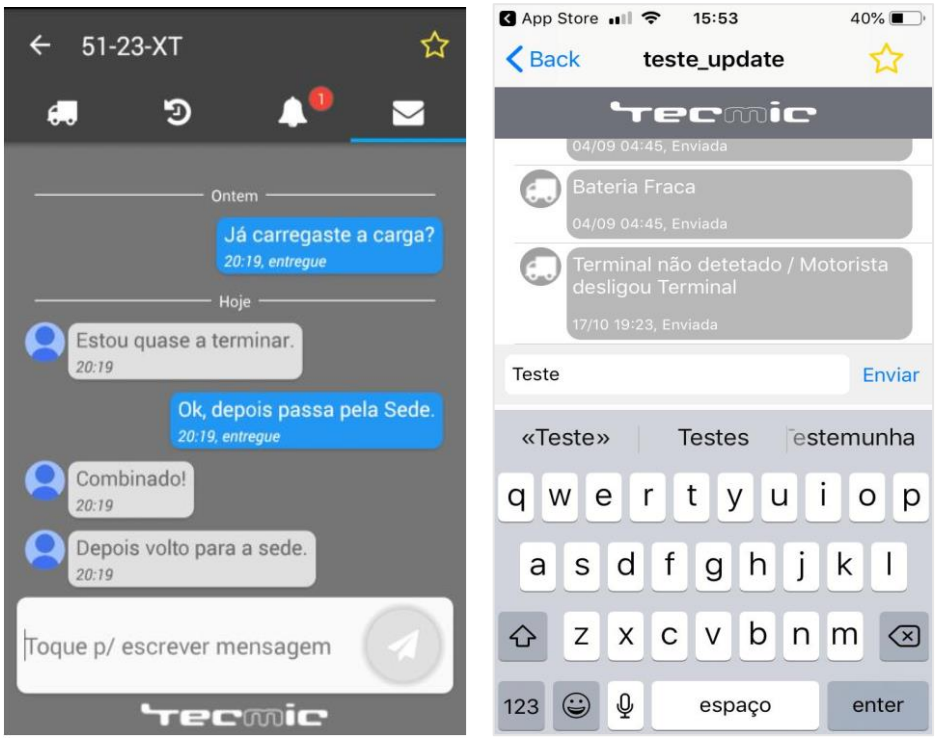


Figura 40 - Caixa de mensagens com um veículo, versão Android (esquerda) iOS (direta);

A **Figura 40** apresenta à esquerda a versão existente Android e à direita a versão criada iOS, este ecrã apresenta o ecrã de mensagens com um veículo. Este ecrã permite receber e enviar mensagens em tempo real entre o utilizador da aplicação e o condutor do veículo, podendo ser visualizada a data e o estado das mesmas (guardada, a enviar, enviada, entregue ou vista). De notar que na versão iOS usou-se o princípio ***Consistency***, tentando ao máximo simular a aplicação de mensagens dos sistemas iOS, conseguindo assim quase uma replica da mesma.

Comparando as duas versões, verificou-se um cuidado acentuado principalmente no tornar este ecrã o mais próximo possível do nativo. Como também na transposição do elemento “Tecmic” da zona de baixo do ecrã para o topo.

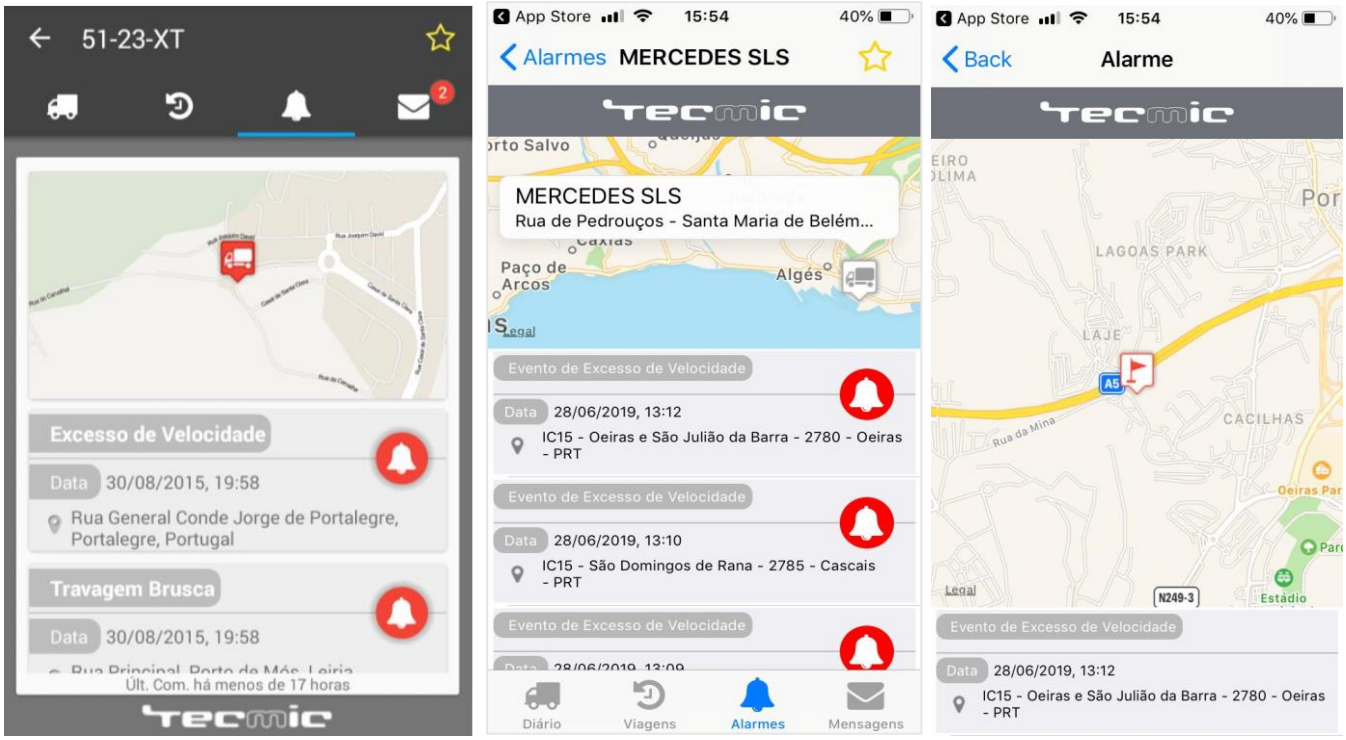


Figura 41 - Ecrã de alarmes de um veículo, versão Android (esquerda) seguida da versão iOS e à direta a representação do alarme em mapa na versão iOS;

A **Figura 41** apresenta à esquerda a versão existente Android e à direita a versão criada iOS, este ecrã permite visualizar todos os alarmes desencadeados pelo veículo selecionado, mostrando o nome do alarme, localização e data de criação, podendo ainda visualizar a posição atual do veículo em mapa, como também a localização em mapa do alarme desencadeado clicando no ícone vermelho.

De notar que tanto o ecrã Alarmes, como o ecrã Mensagens têm um indicador a vermelho, no topo do ícone representativo de cada ecrã, para que este possa indicar ao utilizador o número de alarmes e de mensagens que o veículo selecionado contém que ainda estão por visualizar.[4]

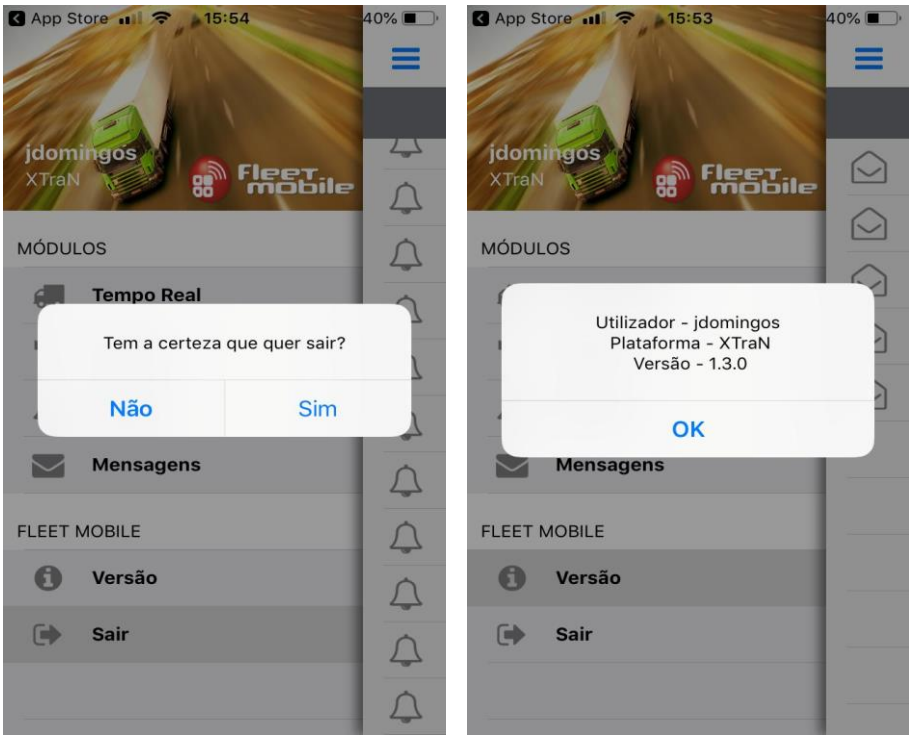


Figura 42 - Alertas sobre ações perigosas e feedback sobre a aplicação;

Na **Figura 42** podemos ver dois exemplos que se baseiam tanto no princípio ***Feedback*** como no princípio ***User Control***.

No exemplo da esquerda podemos ver um alerta sobre uma confirmação de uma operação perigosa, como neste caso o terminar sessão. Enquanto que no exemplo da direita podemos ver um *template* de *feedback* sobre neste caso a versão da aplicação.

Os elementos textuais (escrita) da aplicação seguiram também as diretrizes e *guidelines* iOS, por forma a manter a interface concisa, simples e amigável. De notar que para o desenho de todos os ícones de todos os ecrãs da aplicação foi utilizado a *framework FontAwesome* [20].

Para a visualização de todos os ecrãs da aplicação, consultar os anexos.

Capítulo 5

Avaliação e Testes

Esta secção apresenta os testes que foram realizados, nomeadamente testes unitários, integração, carga, e testes de usabilidade. Maior parte dos testes feitos já tinham sido criados para o desenvolvimento da versão Android nativa desta aplicação, como tal muitos foram reaproveitados e executados neste novo projeto.

Nas subsecções deste capítulo são apresentados os testes que foram realizados para provar conceitos adotados ao longo deste projeto, mas também para garantir que novos desenvolvimentos não danifiquem os existentes, diminuir custos associados à manutenção e aumentar a qualidade do produto desenvolvido [4].

5.1 Testes Unitários

Estes testes têm o intuito de validar o código através da entrada e saída de dados num pequeno e isolado excerto de código. Estes testes foram executados ao longo de cada iteração, para testar cada função desenvolvida validando se retornam o resultado esperado ou não. Para tal foram aproveitados os testes que outrora já tinham sido criados [4], tendo como resultados a **Figura 43** e a **Figura 44**, imagens retiradas da folha de calculo Excel de testes criado.

	Teste	Descrição	Parâmetros	Resultado Esperado	Resultado Real
Veículos	Teste 1	Total de veículos	Lista de Veículos	5	✔
	Teste 2	Total de veículos em movimento	Lista de Veículos	2	✔
	Teste 3	Total de veículos em trânsito	Lista de Veículos	1	✔
	Teste 4	Total de veículos parados	Lista de Veículos	2	✔
	Teste 5	Total de veículos desligados	Lista de Veículos	0	✔
	Teste 6	Condutor ativo	Lista de Condutores	"João Domingos"	✔
	Teste 7	Localização descritiva atual	Veículo	"Aldeamento Santa Clara, Rua da Carvalha, 570, 2400-441 Leiria, Portugal"	✔
	Teste 8	Estado atual	Veículo	"Movimento"	✔
Viagens	Teste 9	Data de partida	Lista de Viagens	"25-09-2015"	✔
	Teste 10	Localização descritiva de partida	Lista de Viagens	"Edifício Ciência II, Av. Prof. Dr. Cavaco Silva, nº13, 3A, 2740-120 Porto Salvo, Portugal"	✔
	Teste 11	Data de chegada	Lista de Viagens	"25-09-2015"	✔

Figura 43 - Primeiros onze testes unitários;

	Teste 12	Localização descritiva de chegada	Lista de Viagens	"Aldeamento Santa Clara, Rua da Carvalha, 570, 2400-441 Leiria, Portugal"	✔
	Teste13	Tempo total em movimento	Lista de Viagens	"01:46:56"	✔
	Teste 14	Tempo total em trânsito	Lista de Viagens	"00:16:32"	✔
	Teste 15	Tempo total parado	Lista de Viagens	"00:05:24"	✔
	Teste 16	Tempo total desligado	Lista de Viagens	"00:00:00"	✔
	Teste 17	Tempo total da viagem	Lista de Viagens	"02:08:52"	✔
	Teste 18	Total de ignições	Lista de Viagens	10	✔
	Teste 19	Total de quilómetros	Lista de Viagens	183	✔
	Teste 20	Média de velocidade	Lista de Viagens	87	✔
	Teste 21	Condutor com mais percursos	Lista de Viagens	"João Domingos"	✔
Alarmes	Teste 22	Último alarme	Lista de Alarmes	"Pânico"	✔
	Teste 23	Total de alarmes não vistos	Lista de Alarmes	0	✔
	Teste 24	Total de alarmes vistos	Lista de Alarmes	2	✔
	Teste 25	Total de alarmes com alta prioridade	Lista de Alarmes	1	✔
Mensagens	Teste 26	Última mensagem	Lista de Mensagens	"Olá"	✔
	Teste 27	Total de mensagens não lidas	Lista de Mensagens	1	✔
	Teste 28	Total de mensagens lidas	Lista de Mensagens	3	✔
	29	Total de mensagens enviadas	Lista de Mensagens	2	✔
	30	Primeiras mensagens de cada dia	Lista de Mensagens	1, 3	✔
	31	Mensagem recebida	Mensagem Recebida	Verdade	✔
	32	Mensagem enviada	Mensagem Enviada	Verdade	✔

Figura 44 - Restantes testes unitários

Os testes unitários, como já foram referidos, foram executados ao longo de cada iteração, divididos basicamente em quatro iterações correspondentes ao término de cada um dos seguintes módulos, "Veículos", "Viagens", "Alarmes" e "Mensagens". Quando cada um destes módulos foi fechado, foram então executados estes testes. Os quais obtiveram o resultado esperado como pode ser visto na ultima coluna da **Figura 43** e da **Figura 44**.

5.2 Testes de Integração

Os testes de integração são testes de software usados com o intuito de testar a integração entre vários módulos e camadas. Nestes testes foram testadas funcionalidades por completo, como por exemplo, o envio de uma mensagem e a respetiva receção da resposta, ou a atualização da velocidade e da posição geográfica com o passar do tempo num veículo com estado "em movimento".

Com este teste foi possível testar tanto as próprias funcionalidade como a *framework SignalR* testando a receção e notificação da resposta. Para estes testes foram utilizados, como já referido anteriormente, os testes que outrora já haviam sido criados. Sendo que também muitas destas funcionalidades foram testadas na própria interface, sem recorrer a testes codificados. Segue de seguida uma figura do teste retirado da referência [4] do envio e receção de mensagens.

```
public class SendMessagesTest {
    @Test
    public final void testRequestSendMessage() {
        //set up params
        MessagesPresenter paramPresenter = new MessagesPresenter();
        TestMessagesView paramView = new TestMessagesView();
        MessagesModuleViewEvent paramRequestSendMessageEvent =
                new RequestSendMessageTestEvent().view(paramView)
                        .identifier(1).descriptionMessage("Hello Driver");
        //expected result
        boolean expectedResult = true;
        //execute
        try {
            synchronized (paramView) {
                paramView.post(paramRequestSendMessageEvent);
                paramView.wait(5000);
            }
        } catch (Exception e) {
            fail(e.getMessage());
        }
        boolean result = paramView.result;
        //assert result
        assertEquals(expectedResult, result);
    }
    private static class TestMessagesView implements SendMessagesTestAPI {
        protected boolean result;
        @Override
        public void responseMessage(boolean response) {
            result = response;
            synchronized (this) {
                notifyAll();
            }
        }
        public void post(MessagesModuleViewEvent messagesModuleEvent) {
            MessagesBroadcast.getInstance().post(messagesModuleEvent);
        }
    }
}
```

SendMessagesTest (pt.tecmic.modules.messages)
All Tests Passed

Figura 45 - Teste de integração assíncrono para envio de mensagem e resultado do mesmo; [4]

5.3 Testes de Carga

Os testes de carga foram também bastante importantes, imaginando que há clientes com frotas com milhares de carros, obter por exemplo as conversas destes veículos pode ser uma operação bastante dispendiosa caso não seja implementada de forma exemplar. Como tal foram feitos testes nesta ótica. Seguindo uma imagem da tabela do ficheiro de Excel dos resultados na **Figura 46**.

	Número de veiculos						
	10	20	50	100	500	1000	10000
Renderizar Veículos no Mapa	≈1seg	≈1seg	≈1seg	≈2seg	≈5seg	≈10seg	≈22seg
Carregar Lista Viagens	≈1seg	≈1seg	≈3seg	≈12seg	≈30seg	≈62seg	≈170seg
Carregar Lista Alarmes	≈1seg	≈1seg	≈1seg	≈2seg	≈5seg	≈16seg	≈93seg
Carregar Lista Mensagens	≈1seg	≈5seg	≈15seg	≈24seg	≈41seg	≈93seg	≈218seg

Figura 46 - Resultados dos testes de carga;

Não foram executados testes sobre o volume de dados que são transferidos entre o servidor e a aplicação móvel pois estes já tinham sido feitos na primeira versão. Os resultados não foram nada satisfatórios, como tal procedeu-se à otimização do lado servidor para este fornecer só os dados necessários e não todos os dados referentes ás entidades necessárias, obtendo assim tempos praticamente instantâneos.

5.4 Testes de Usabilidade

Ao longo do desenvolvimento desta aplicação foram planeados e executados vários testes de usabilidade, numa primeira fase a um grupo de funcionários de um cliente da Tecmic S.A. (10 pessoas). E numa segunda fase a utilizadores mais avançados, grupo selecionado de funcionários da empresa (15 pessoas), pois apresentam um forte conhecimento sobre o modelo de negócio onde a aplicação se enquadra. Foram escritos então vários guiões de tarefas para os mesmos executarem.

1. Inicie a sessão com nome de utilizador: “bcarola” e palavra-passe: “teste123”.
2. Quantos veículos se encontram em Movimento?
3. Selecione o veículo com o seguinte nome: “Mercedes SLS”.
4. O veículo selecionado encontra-se em movimento?
5. Tem algum condutor? Se sim, qual?
6. Pesquise o veículo com o seguinte nome: “Mercedes S400 Híbrido”.
7. Coloque o veículo pesquisado como favorito.
8. Qual foi o último alarme que o veículo selecionado desencadeou?
9. Envie uma mensagem com o texto “Viagem ao cliente X cancelada” para o veículo com o nome: “Mercedes SLS”.

Estes testes foram parcialmente em vão, pois, todos os utilizadores desta aplicação já apresentava uma enorme experiência sobre a versão Android nativa desta aplicação. Como tal, os resultados foram 100% bem sucedidos.

Por motivos de confidencialidade não foi possível arranjar uma amostra de pessoas externas ao projeto.

5.5 Discussão de resultados

Nesta secção será discutido todos os resultados dos testes feitos referidos nas secções anteriores, numa forma geral os resultados foram bastante positivos, com exceção dos testes de carga.

Com os testes unitários foi possível validar excertos pequenos e isolados de código das várias funcionalidades procurando garantir a coesão e a fiabilidade do código das mesmas. Como mostrado na **Figura 43** e **Figura 44**, o sistema passou com exatidão em todos os testes.

Em relação aos testes de integração, mais concretamente o da **Figura 45** com o envio e receção de mensagens, foi possível aferir que a integração entre todas as camadas referidas na **secção 4.5.1** – camadas estas que dizem respeito á integração entre o *Mvvm-Cross*, *Xamarin* e a *framework ASP.Net* – foi bem sucedida. Como também foi possível testar a integração com a *framework SignalR* na receção da mensagem.

Com base nos resultados dos testes de carga, **Figura 46** - Resultados dos testes de carga; foi possível aferir que o carregamento de maior parte dos ecrãs de listas – lista de mensagens, lista de viagens ou mesmo a lista de alarmes – quando o número de veículos da frota escala para as centenas e principalmente para os milhares apresentava tempos de carregamentos excessivos quando o objetivo é ter uma utilização fluida da aplicação. Como tal procedeu-se á otimização das camadas de acesso a dados por forma a trazer só os campos que são visíveis na interface, apresentando assim valores praticamente instantâneos de carregamento.

Por fim, como foi referido na **secção 5.4**, tanto a primeira fase como a segunda fase de testes, foram em vão pois todas as amostras de pessoas que foram sujeitas aos teste, já apresentavam uma enorme experiência sobre a versão Android nativa. Como tal, os resultados foram 100% bem sucedidos, sendo possível aferir que todos os testes feitos e o desenvolvimento de toda a interface da aplicação nativa, foi feita corretamente apresentando níveis de usabilidade bastante fortes.

Importante referir também que para além destes testes, no momento de submissão da aplicação nas várias lojas de cada plataforma (Apple – AppleStore, Android – Play Store), principalmente na Apple Store, a aplicação é sujeita a vários testes intensivos durante no mínimo dois dias por parte da Apple, sendo que esta aplicação passou com distinção todos os testes encontrando-se já disponível na Apple Store.

Capítulo 6

Conclusão e Trabalhos Futuros

Com o intuito de finalizar o presente documento será sintetizado nesta secção todas as conclusões sobre todo o projeto, será discutido também a concretização de cada objetivo e por fim será apresentado também todo o trabalho futuro que se prevê ainda ser desenvolvido.

6.1 Conclusões

Este projeto endereçou sobre a necessidade dos clientes da empresa Tecmic S.A. terem uma versão da aplicação móvel FleetMobile para os seus dispositivos iOS, esta contém a apresentação de veículos em mapa, viagens, alarmes e mensagens com suporte a comunicações em tempo real.

Para o desenvolvimento da mesma foram detalhados requisitos para serem estudados e cumpridos desde quais as melhores ferramentas para o desenvolvimento multiplataforma, estudo do uso das guidelines de iOS às interfaces já desenvolvidas em Android, qual a melhor tecnologia para comunicação em tempo real e qual o melhor padrão de desenho e técnicas para tornar o código reutilizável e que facilite futuras ações de manutenção e atualização de forma uniforme para os dois sistemas alvo de interesse (Android e iOS).

Tal como apresentado neste documento, foi estudado tecnologias, *frameworks* e padrões de desenho que permitissem o desenvolvimento de uma aplicação multiplataforma. Foi então escolhido, na fase de desenvolvimento o *Xamarin* e a *framework Mvvm-Cross* para o desenvolvimento multiplataforma com o auxílio da *framework SignalR* para as funcionalidades em tempo real da aplicação. Conseguiu-se uma arquitetura escalável que dá resposta às necessidades do cliente, como também possibilita futuras adições no processo de negócio do mesmo. Esta arquitetura veio concretizar os objetivos pretendidos de ter um sistema com o código reutilizável e que facilite futuras ações de manutenção e atualização de forma uniforme para as duas plataformas. Para além da arquitetura desenhada, foi possível também – com base na versão nativa android desta aplicação – ter uma interface iOS que seguisse as guidelines iOS sem que houvesse grandes alterações.

Deste projeto resultou então a publicação e disponibilização da aplicação FleetMobile para os sistemas iOS, sendo este o principal requisito deste projeto. Conseguindo assim colmatar a necessidade dos clientes da empresa de ter uma versão iOS da aplicação e em simultâneo cumpriu-se todos os objetivos detalhados pela empresa antes do desenvolvimento da mesma. No entanto o sistema apresenta potencial de evolução e necessidade de trabalho futuro.

6.2 Trabalhos Futuros

Como referido na secção anterior, este projeto tem um grande potencial de evolução, no entanto existe a necessidade de trabalho futuro. É neste último ponto que esta secção irá focar-se com o seguinte enumerado, sendo que os primeiros pontos vêm ainda da primeira versão nativa [4]:

- A adição de mais módulos, como o de eficiência energética, de tempos de condução e de pontuação de condutores, usando o conceito de *Gamification* [4];
- A adição de comunicações por voz para os veículos da frota através da API de voz do dispositivo móvel. Esta adição não foi considerada prioritária porque muitos dos veículos ainda não possuem comunicação por voz através da consola gráfica Tecmic [4];
- A identificação das ações que a aplicação pode fazer offline, para que se proceda à implementação de uma base de dados na aplicação móvel. Esta adição não foi considerada prioritária uma vez que o grande objetivo da aplicação é disponibilizar informação em tempo real, como localizações, alarmes, mensagens, entre outros [4];

De seguida serão numerados os pontos provenientes do desenvolvimento desta versão:

- Melhorar alguns detalhes do desenho das interfaces da aplicação;
- Finalizar o desenvolvimento da versão Android neste sistema multiplataforma por consequência fazer todos os testes necessários e a sua análise. Este ponto não foi prioritário pois o principal requisito era a disponibilização da versão iOS;
- Disponibilizar a versão Android na loja aplicacional do Android – *Play Store*;

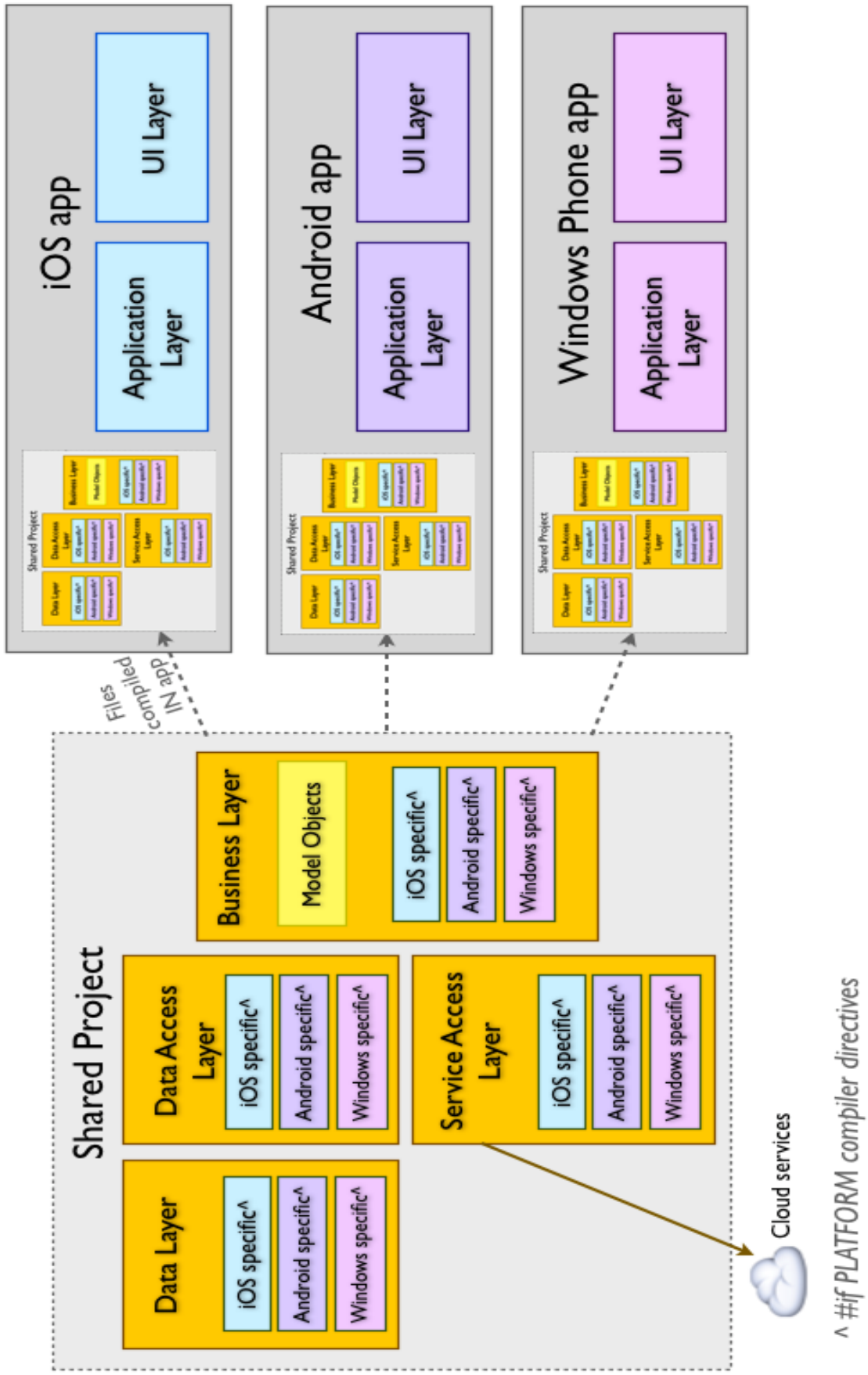
Referências

1. USWITCH, History of mobile phones and the first mobile phone - https://www.uswitch.com/mobiles/guides/history-of-mobile-phones/ [acedido Out 6 2018]
2. Alberto Furlan, Cross Platform Mobile App Development Guide (2017), [acedido Nov 10 2018]
3. Anirudh Nagesh & Carlos E. Caicedo, *(PDF) Cross-Platform Mobile Application Development*. Available from: https://www.researchgate.net/publication/263416908_Cross-Platform_Mobile_Application_Development [acedido Nov 29 2018].
4. João Miguel Henriques Domingos – Fleet Mobile Managment – Relatório de Estágio – 2015. [acedido Out 7 2019]
5. Android, Android Studio, https://developer.android.com/studio/. [acedido Out 7 2019]
6. Triet Le, iOS Development with Swift programming language, Available from https://www.theseus.fi/bitstream/handle/10024/124255/Triet-Le-Thesis-Final-05042017.pdf?sequence=1 [acedido Out 7 2019]
7. Matteo Manferdini, iOS Storyboards in Xcode: The Ultimate Guide, https://matteomanferdini.com/ios-storyboards-xcode/ [acedido Jul 26 2019]
8. Apple, XCODE 10, https://developer.apple.com/xcode/ [acedido Jul 26 2019]
9. Mukesh Prajapati & Dhananjay Phadake & Archit Poddar, STUDY ON XAMARIN CROSS-PLATFORM FRAMEWORK, https://www.academia.edu/37016885/Study_on_Cross_-Platform_Mobile_App_Development_With_Xamarin [acedido Jul 26 2019]
10. Craig Dunn & Brad Umbaugh, Sharing code overview, https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/code-sharing . [acedido Aug 20 2019]
11. ANUPAM CHUGH, MVVM Design Pattern, https://www.journaldev.com/20292/android-mvvm-design-pattern [acedido Aug 22 2019]
12. Neeraj Agarwal, Introduction To PhoneGap, https://www.formget.com/phonegap/ [acedido Out 06 2019]
13. John Dooley. Software Development and Professional Practice. Apress, 2011. [acedido Set 06 2019]
14. DevMedia, Introdução ao Padrão MVC. [acedido Set 06 2019]
15. Techbrij, SignalR, https://techbrij.com/database-change-notifications-asp-net-signalr-sqldependency . [acedido Set 06 2019]
16. Statcounter Global Stats, mercado iOS, http://gs.statcounter.com/ios-version-market-share/mobile-tablet/worldwide . [acedido Set 06 2019]
17. Gill Oliver, Using React Native for mobile software development, https://www.theseus.fi/bitstream/handle/10024/143282/Gill_Oliver.pdf?sequence=1&isAllowed=y . [acedido Out 10 2019]
18. AltexSoft, Inc [US], The Good and The Bad of Xamarin Mobile Development, https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/ [acedido Out 10 2019]
19. Apple, Human Interface Guidelines, https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/ [accessed Out 11 2019]
20. FontAwesome, https://fontawesome.com [acedido Out 11 2019]

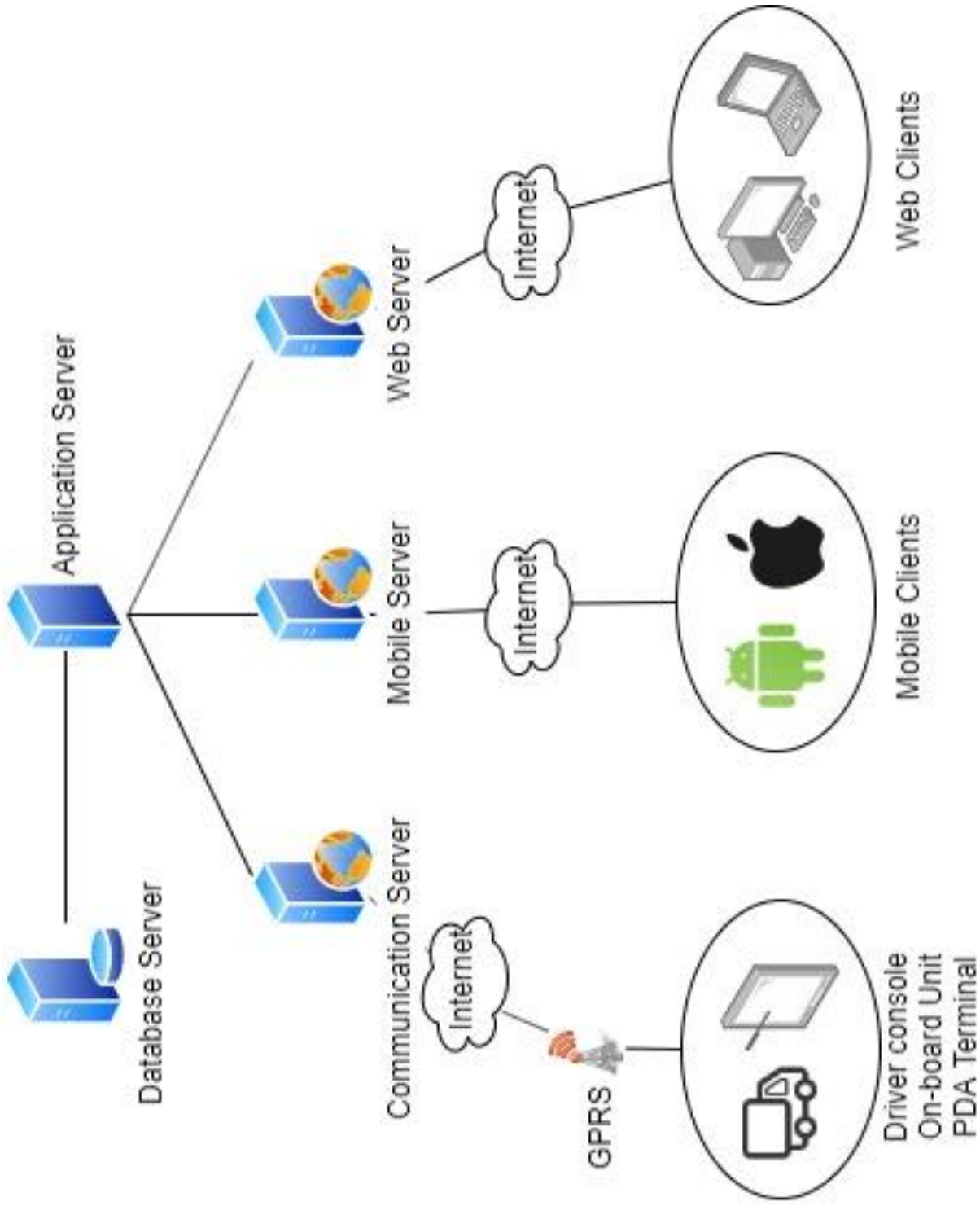
Anexos

Anexo 1 – Arquitetura .NET Standard Library – Figura 4

Anexo 2 – Arquitetura Shared Projects – Figura 5

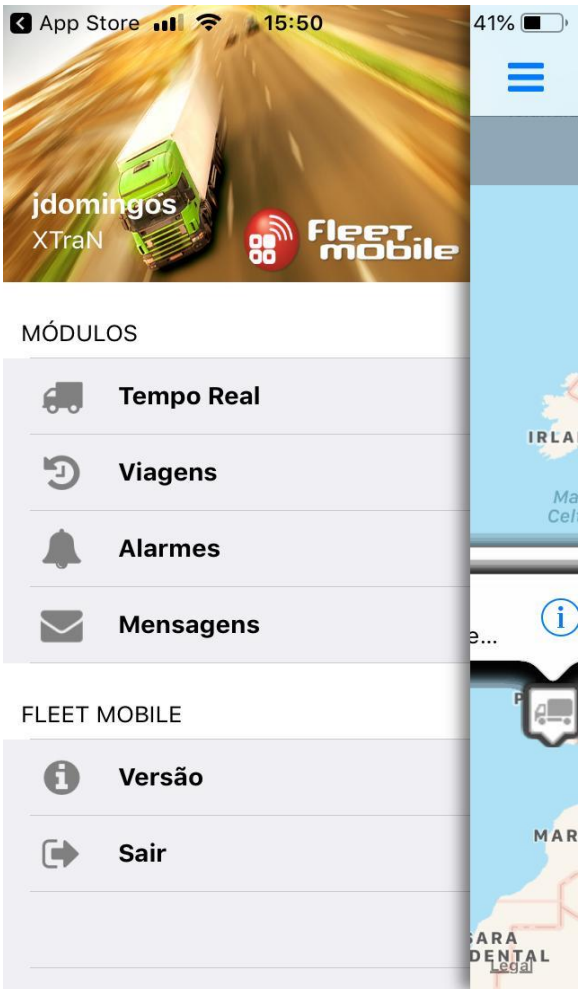


Anexo 3 – Arquitetura da infraestrutura do sistema iZiTraN – Figura 14

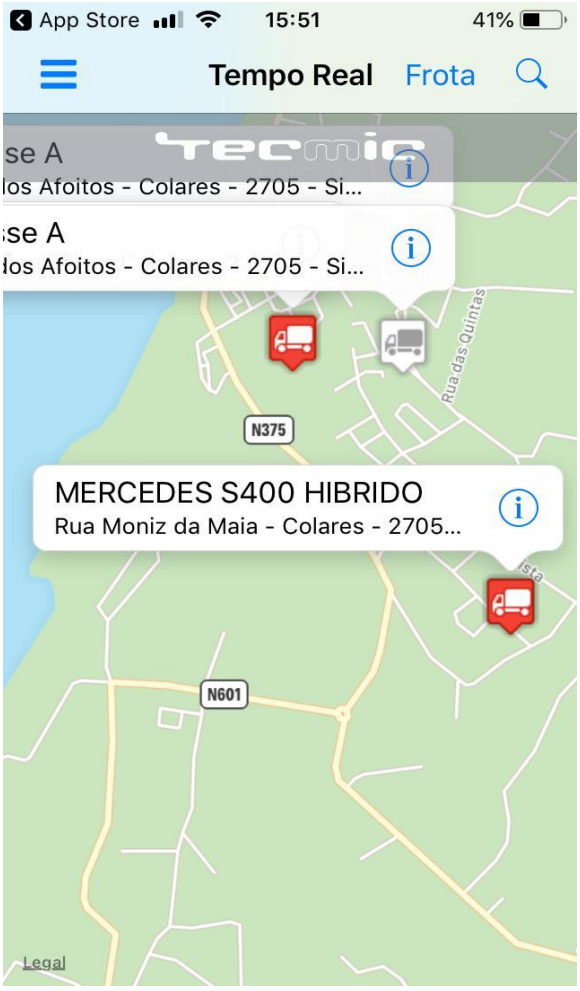


Anexo 4 – Capturas de Ecrã de todas as Interfaces da Aplicação iOS

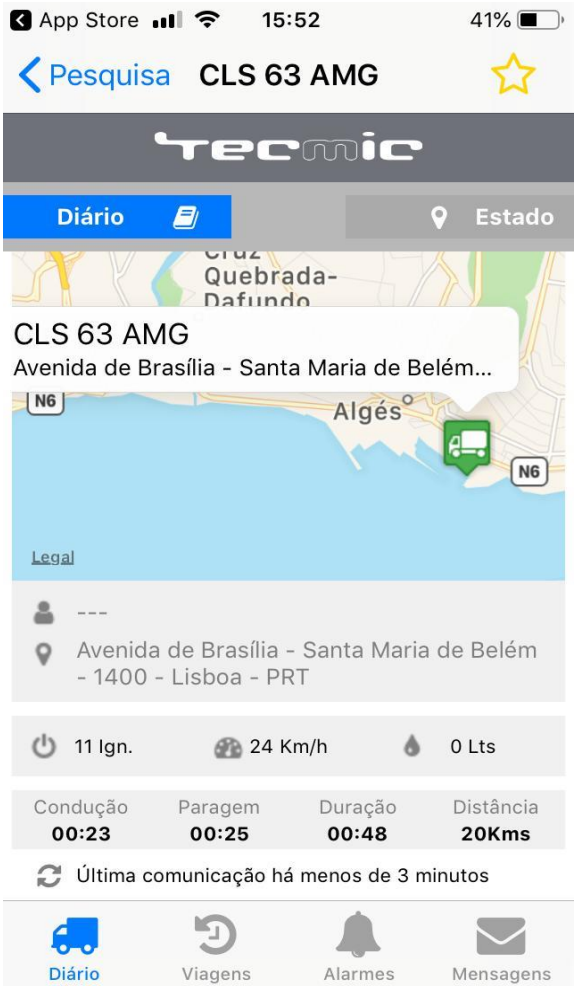
1 – Ecrã de Login;



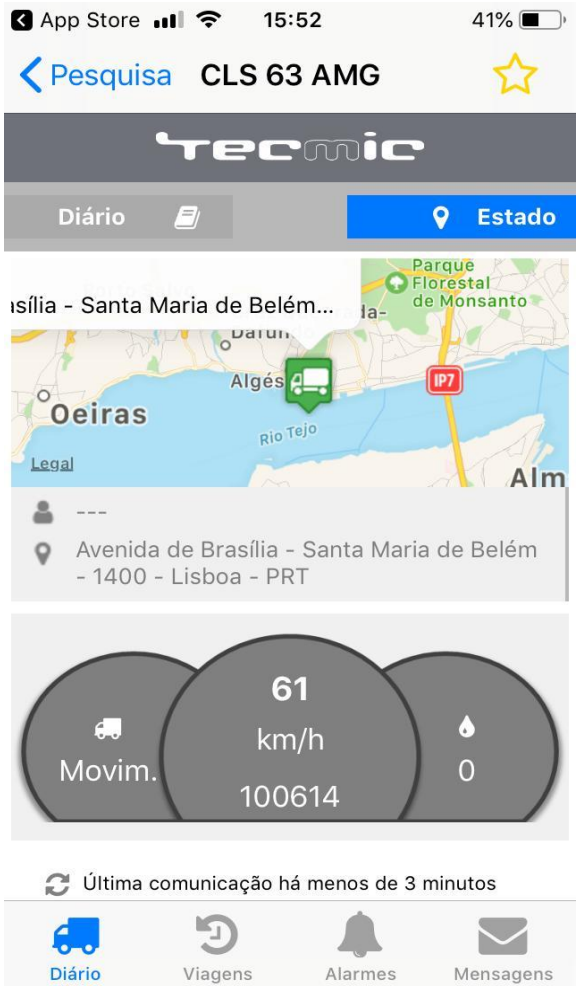
2 – Menu de Navegação Lateral;



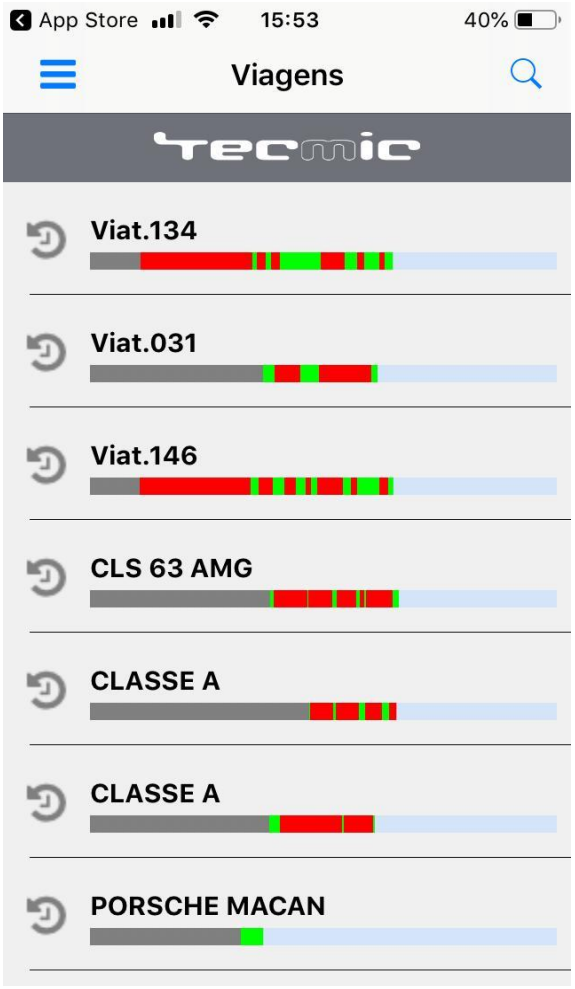
3 – Ecrã de Tempo Real;



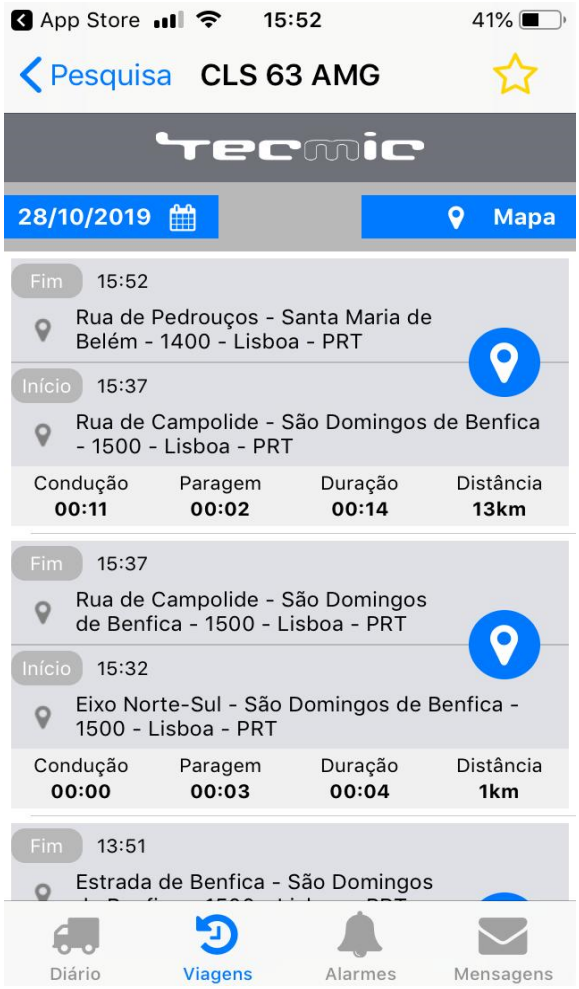
4 – Ecrã Diário do Veículo;



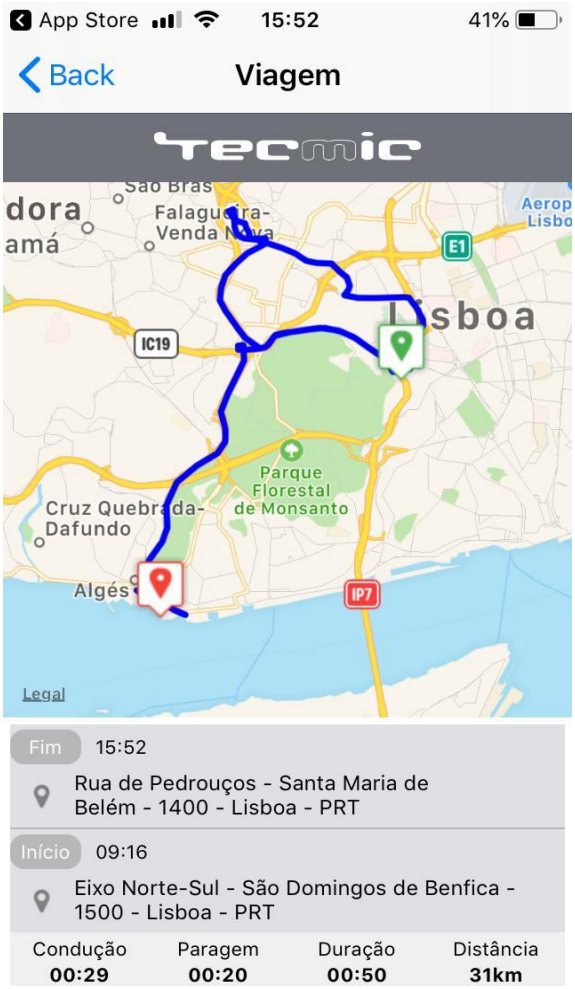
5 – Ecrã Estado do Veículo;



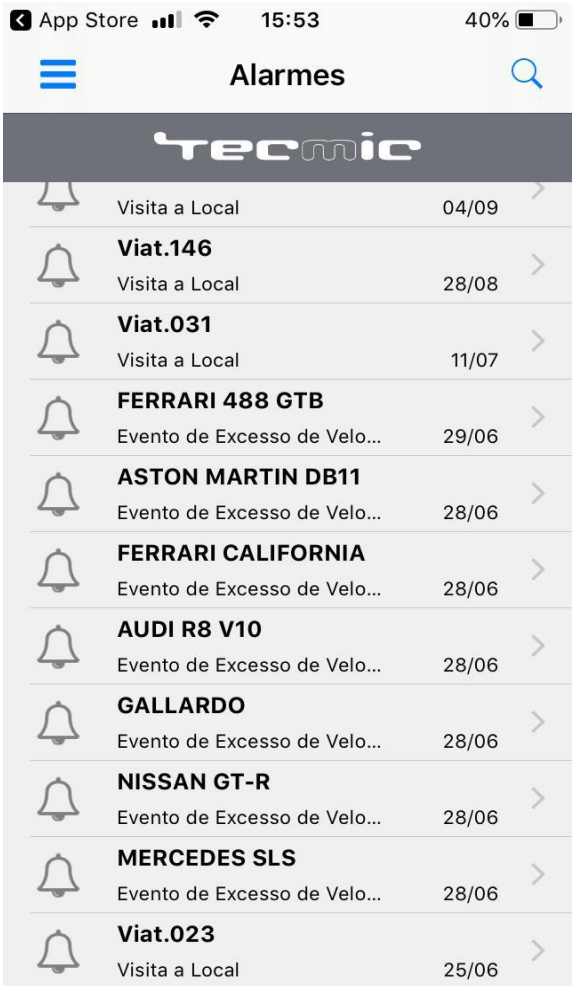
6 – Lista de viagens de cada veículo;



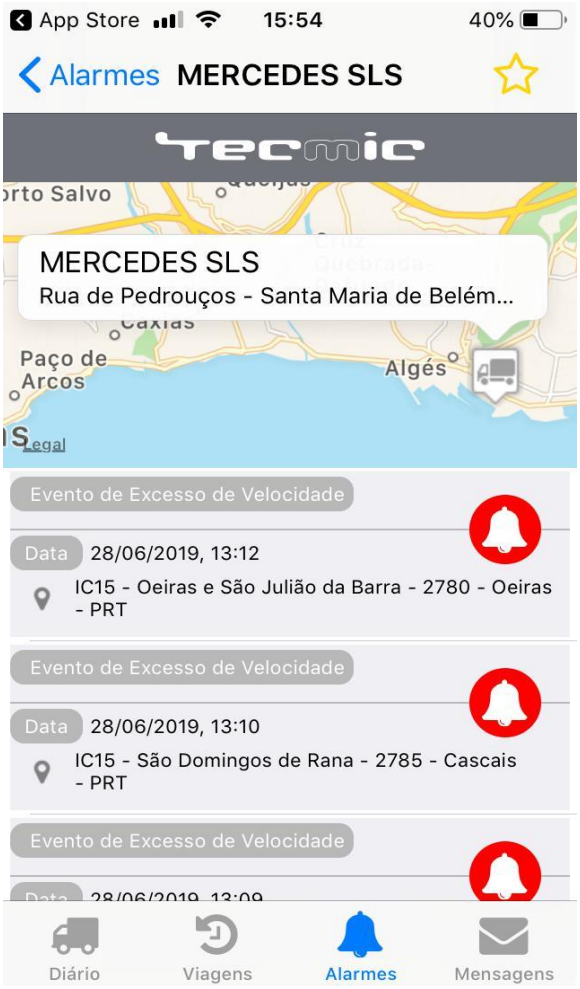
7 –Lista de viagens do veículo;



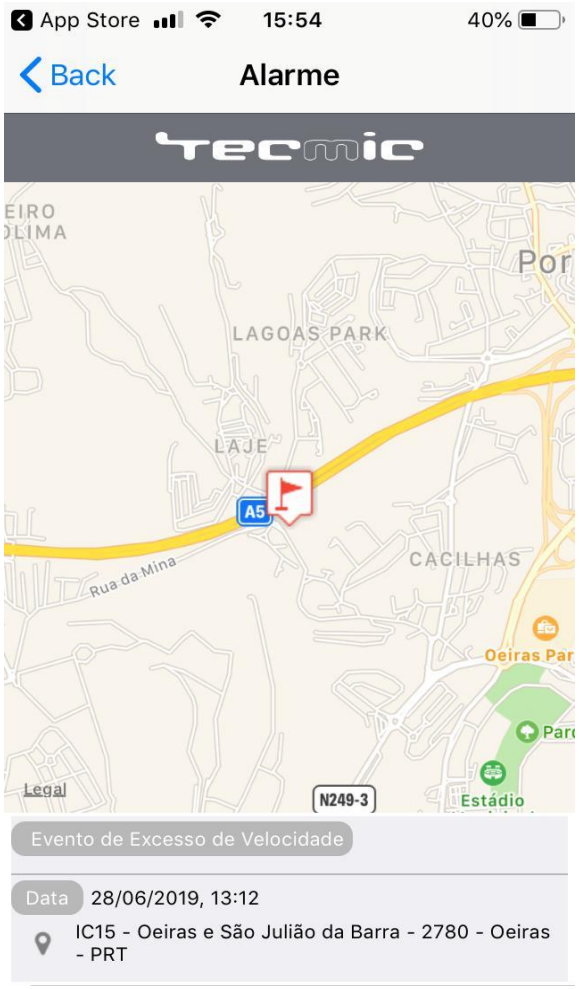
8 – Representação de uma viagem;



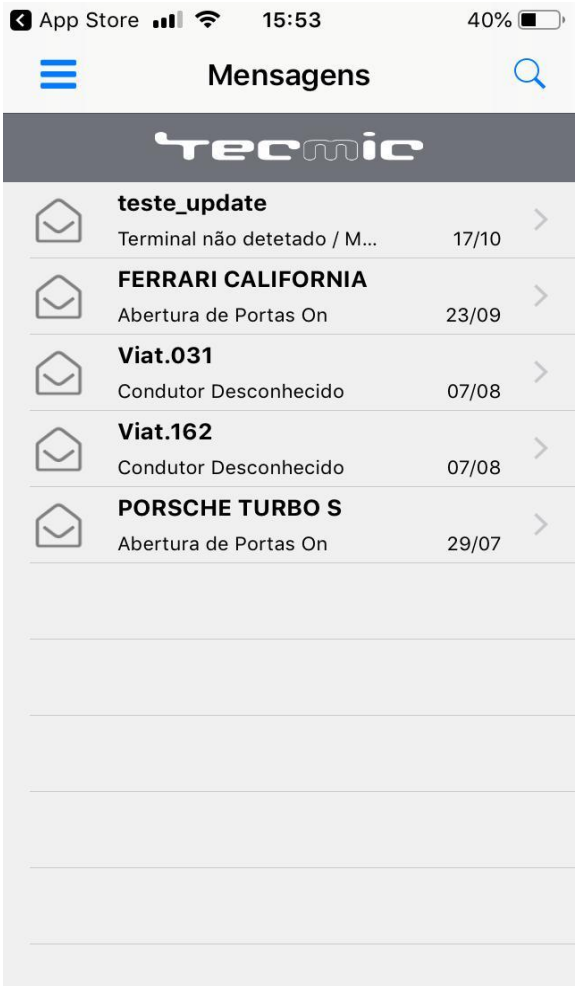
9 – Lista de alarmes de cada veículo;



10 – Lista de alarmes do veiculo;



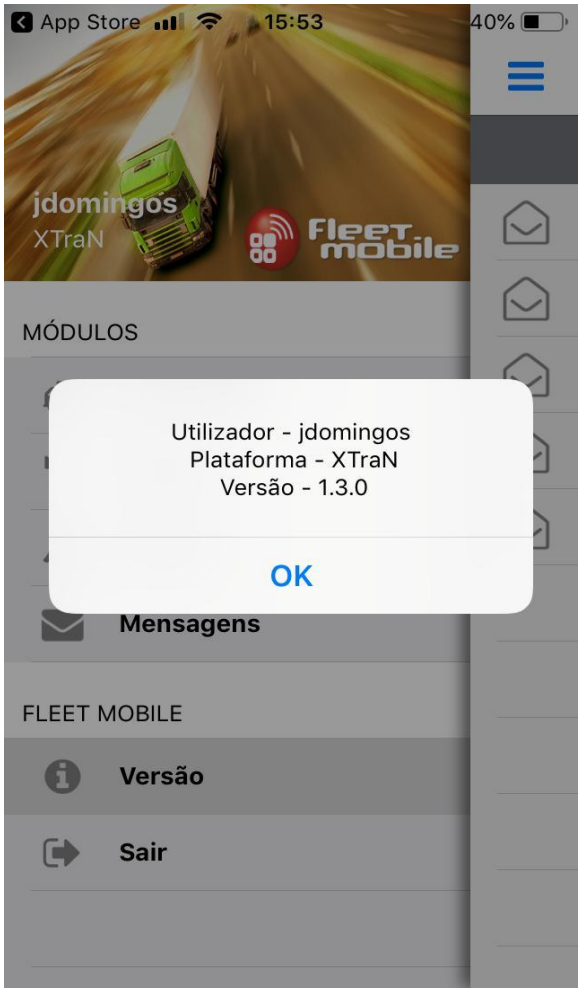
11 – Representação de um alarme;



12 – Lista de conversas;

13 – Ecrã de conversa com veículo;

14 – Ecrã de pesquisa de veículos;



15 – Ecrã de informação sobre aplicação;

16 – Alerta de operação perigosa;