Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Eng.ª Informática – Computação Móvel

XTRAN PASSENGER BUS MONITOR MOBILE APP

TIAGO FRANCISCO DIAS PARENTE

Leiria, Setembro de 2022

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Eng.ª Informática – Computação Móvel

XTRAN PASSENGER BUS MONITOR MOBILE APP

TIAGO FRANCISCO DIAS PARENTE
Número: 2202291

Estágio realizado sob orientação do Professor Doutor José Ribeiro (jose.ribeiro@ipleiria.pt).

Leiria, Setembro de 2022

AGRADECIMENTOS

Ao longo da realização deste projeto tive o apoio de várias pessoas.

Em primeiro lugar, gostaria de agradecer ao professor José Ribeiro pelo apoio prestado ao longo deste ano.

Quero também agradecer ao Joni Batista, António Marcelo e ao Pedro Silva, que foram os meus orientadores na Tecmic, e também aos meus colegas de trabalho Joel Fernandes, André Leal, André Sousa e Pedro Rino que sempre me apoiaram no trabalho quando foi necessário.

Por fim, agradeço a toda a minha família pelo apoio ao longo de todo o mestrado, e aos meus colegas Henrique Santos, Francisco Fernandes e João Borges por estarem sempre disponíveis para ajudar em qualquer coisa que fosse necessária.

RESUMO

Hoje em dia as empresas necessitam cada vez mais de uma forma simples de analisar dados para reduzir custos de certos processos de negócio, bem como otimizar os mesmos. No caso do projeto apresentado neste documento, realizado no âmbito da unidade curricular de Estágio, na empresa Tecmic S.A., foi criada uma solução que otimiza o processo de gestão e análise do desempenho de motoristas, bem como a eficiência da condução dos mesmos, com o intuito de reduzir custos no processo de condução de autocarros.

O objetivo deste projeto consistiu no desenvolvimento de uma aplicação móvel para Android e iOS (recorrendo a uma tecnologia que permita desenvolvimento multi-plataforma), bem como o desenvolvimento da infraestrutura necessária para o funcionamento da mesma. Neste caso, isto consistiu na criação de numa API de autenticação, uma API de dados de eficiência energética, e uma API Gateway.

A aplicação móvel permite que gestores e motoristas de autocarros possam visualizar o seu desempenho ao longo do tempo, em que aspetos estes podem melhorar e otimizar a sua condução, e onde estes cometeram erros ao longo dos percursos feitos. Esta aplicação incentiva uma condução mais otimizada para os motoristas, e introduz uma nova plataforma para os gestores poderem visualizar o desempenho dos seus diversos motoristas.

ABSTRACT

Nowadays, companies increasingly need a simple way to analyze data to reduce costs of certain business processes, as well as optimize them. In the case of the project presented in this document, carried out as part of the Internship course unit, at Tecmic S.A., a solution was created that optimizes the process of managing and analyzing the performance of drivers, as well as their driving efficiency, with the aim of reducing costs in the process of driving buses.

The goal of this project was to develop a mobile application for Android and iOS (using a multi-platform framework), as well as to develop the necessary infrastructure to make it work. In this case, this consisted in the creation of an authentication API, an energy efficiency data API, and an API Gateway .

The mobile application allows bus managers and drivers to view their performance over time, where they can improve and optimize their driving, and where they have made mistakes along the way. This application encourages more optimal driving for drivers, and introduces a new platform for managers to view the performance of their various drivers.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	Application Programming Interface.
BD	Base de dados.
CD	Continuous Delivery.
CI	Continuous Integration.
CSS	Cascading Style Sheets.
DTO	Data Transfer Oject.
ECDSA	Elliptic Curve Digital Signature Algorithm.
GPS	Global Positioning System.
GUI	Graphical User Interface.
HMAC	Hash-Based Message Authentication Code.
HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.
IIS	Internet Information Services.
IOC	Inversion Of Control.
IP	Internet Protocol.
JSON	JavaScript Object Notation.
JWT	JSON Web Token.
MVVM	Model-View-ViewModel.

ORM	Object–relational mapping.
RBAC	Role-Based Access Control.
RFC	Request for Comments.
RSA	Rivest-Shamir-Adleman.
SDK	Software Development Kit.
UI	User Interface.
UM	Unidade móvel.
URL	Uniform Resource Locator.
UUID	Universally Unique Identifier.
XAML	Extensible Application Markup Language.

1

INTRODUÇÃO

Atualmente as empresas necessitam cada vez mais de uma forma de analisar dados para tornar os processos de negócio mais eficientes, reduzindo os custos destes. Muitas vezes é também necessário que os trabalhadores tenham acesso ao seu desempenho, para que estes tenham a informação necessária de onde podem melhorar. Para isto tanto os gestores de uma empresa como os trabalhadores necessitam de acesso a uma ferramenta que contenha estes dados, sendo as principais plataformas para isto aplicações *web* e aplicações móveis. Isto aplica-se a qualquer área de negócio, no entanto, este estágio foca-se na área dos transportes públicos de passageiros, mais especificamente na criação de uma aplicação móvel multi-plataforma (Android e iOS), para gestores de frotas e motoristas. A aplicação foca-se na área de eficiência energética, e visa reduzir custos de combustível, ao incentivar uma condução mais eficiente, levando assim à redução dos custos de manutenção de veículos, como consequência disto.

Apesar de o desenvolvimento desta aplicação móvel multi-plataforma ser o principal objetivo deste estágio, para que esta tenha acesso a dados foi também necessário desacoplar uma API (Application Programming Interface) que disponibilizasse dados de eficiência energética de um monólito que esta empresa tem, conhecido por XtpServer. Foi necessário criar um serviço de autenticação para a aplicação móvel, que fosse escalável de maneira a que no futuro este possa ser utilizado por todo o tipo de serviços desacoplados do XtpServer. Visto que a empresa planeia evoluir gradualmente para uma arquitetura de micro-serviços, foi também pensado numa arquitetura que fosse escalável.

É de notar que apesar da aplicação móvel inicialmente ter o nome de *XTraN Passenger Bus Monitor Mobile App*, ao longo do desenvolvimento do projeto, esta foi renomeada para *XtpDriveMobile* (sendo este o nome pelo qual é feita referencia à aplicação ao longo deste documento), pois foi criado todo um novo sistema como o nome *XtpDrive*.

Alguns dos pressupostos deste estágio foram as tecnologias utilizadas. Para a aplicação móvel foi utilizado Xamarin Forms, e para as API foi utilizado .NET, pois a empresa contém a maioria dos serviços implementados com ferramentas descendentes de C# e .NET. Outro fator para isto é a maioria dos profissionais serem especializados nestas tecnologias, facilitando assim que estes possam alternar entre projetos mais facilmente.

1.1 CARACTERIZAÇÃO DA ENTIDADE DE ACOLHIMENTO

A entidade de acolhimento do estágio realizado foi a Tecmic S.A, uma empresa portuguesa fundada em 1988. A Tecmic é uma empresa que opera na área de sistemas de informação e desenvolvimento de software e hardware. Esta contém diversos produtos em diferentes áreas de negócio, sendo estas, por exemplo, fornecer soluções para gestão de frotas de veículos, sistemas de gestão da logística e transporte de mercadorias, sistemas de comando e controle de Forças de Segurança e Emergência, entre outros. A Tecmic atualmente tem clientes em diversos países, nomeadamente Portugal e Brasil, estando entre estes clientes a Carris, NOVAFAOL, Galo Branco, etc.

1.1.1 *Enquadramento do Estágio*

A Tecmic contém diversos projetos e produtos em diferentes áreas de negócio. Este estágio enquadra-se na área do projeto *XTraN Passenger*. Este é um projeto de gestão de frotas, mais especificamente na área de transportes públicos de passageiros.

Este projeto contém diversos componentes, desde gestão de frotas em tempo real com auxílio de uma aplicação web, como também a recolha de dados através dos sistemas embutidos nos diferentes veículos, e ainda a aplicação móvel de passageiros (utilizada atualmente pela Carris).

Uma das principais funcionalidades deste projeto é a criação de relatórios mensais, semanais ou diários sobre a eficiência energética (dados acerca de consumo dos veículos, eficiência de condução dos motoristas, etc). Estes dados podem ser orientados a diferentes entidades, tais como veículos, linhas (rotas) e condutores. É aqui que enquadra o estágio, sendo o principal objetivo criar uma aplicação móvel multi-plataforma (Android e iOS) que irá mostrar estes dados aos utilizadores finais.

Os utilizadores finais desta aplicação são condutores (para consultarem a sua performance) e gestores/administradores de frotas (para visualizarem o desempenho dos diversos trabalhadores). A principal motivação para o desenvolvimento desta aplicação foi o interesse por parte dos clientes da empresa ao longo dos anos, para uma aplicação de *back office* que não só facilitasse os gestores, mas também desse uma boa perspetiva aos condutores do seu desempenho e de como estes podem melhorar.

1.2 ESTRUTURA DO DOCUMENTO

O que resta deste documento está organizado da seguinte forma:

- Os Conceitos e Contextualização são apresentados no capítulo 2, onde é apresentado o *XTraN Passenger Web* e feita uma análise das diferentes tecnologias e abordagens utilizadas ao longo do projeto desenvolvido.
- As Metodologias são apresentadas no capítulo 3, no qual é apresentada a metodologia de desenvolvimento utilizada, bem como as ferramentas que o permitiram.
- Os Requisitos e Arquitetura são apresentados no capítulo 4, onde é dada uma visão geral da arquitetura dos diferentes sistemas envolvidos neste projeto.
- O Trabalho Desenvolvido é apresentado no capítulo 5, no qual são apresentados os diferentes componentes desenvolvidos neste projeto, enumerados de seguida:
  - *XtpDriveAPI*
  - *XtpAuthAPI*
  - *XtpDriveDtos*
  - API Gateway
  - *XtpDriveMobile* (inicialmente conhecido por *XTraN Passenger Bus Monitor Mobile App*)
- Os Testes são apresentados no capítulo 6, onde são especificados os testes realizados para validar o projeto desenvolvido.
- As Conclusões e Trabalho Futuro são apresentadas no capítulo 7, no qual são dadas a conhecer as principais contribuições do projeto desenvolvido e o trabalho que ainda é necessário implementar futuramente.

2

CONCEITOS E CONTEXTUALIZAÇÃO

No desenvolvimento deste projeto foram utilizadas diversas tecnologias e abordagens. Para este efeito foi feita uma análise de tecnologias utilizadas, bem como as suas alternativas. Foram também investigadas outras aplicações móveis no mesmo contexto, no entanto, não foram encontradas aplicações que seja relevante mencionar, visto que a aplicação desenvolvida é uma aplicação de *back office*. Apesar disto, é apresentada a aplicação *web XTraN Passenger Web*, um dos projetos da Tecmic que serviu como base para aplicação móvel desenvolvida.

Neste capítulo é primeiramente apresentada a aplicação *web XTraN Passenger Web*, sendo de seguida apresentadas abordagens de desenvolvimento mobile multiplatforma; depois é apresentada uma visão geral do Xamarin e Xamarin Forms, e por fim é feita uma síntese das diferentes abordagens de autenticação, nomeadamente focando em cenários com micro-serviços.

2.1 XTRAN PASSENGER WEB

O *XTraN Passenger Web* é um dos projetos da Tecmic, que consiste numa aplicação *web* para a gestão de transportes públicos de passageiros. Esta tem diversas funcionalidades, entre as quais a monitorização dos transportes públicos em tempo real e a elaboração e exportação de relatórios de diversos tipos. Com estes relatórios é possível obter várias informações do histórico de viagens.

Um destes relatórios é o relatório de eficiência energética, conhecido por *EcoDriver*, que contêm dados acerca do consumo dos veículos, os excessos cometidos pelos condutores, onde estes excessos foram cometidos e o desempenho do condutor, com um *score* de 0-10. Este tipo de relatórios é bastante importante para refletir nas viagens feitas, bem como detetar onde é possível melhorar para uma condução mais eficiente, reduzindo assim o consumo médio da empresa.

Na Figura 1 pode-se ver um exemplo de um relatório de eficiência energética. Este contém vários filtros possíveis, sendo estes o intervalo de tempo, a qualidade dos dados, considerar apenas dados em viagem ou não, agrupar dados por mês, semana, dia ou globalmente, e orientar os dados a uma entidade (motorista, linha, veículo).

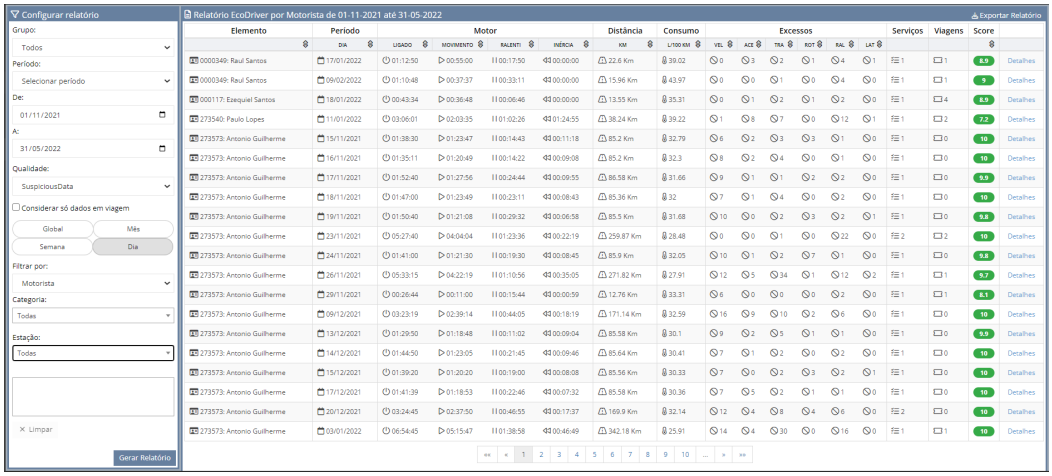


Figura 1: Ecrã de um relatório de eficiência energética

Nos dados resultantes podemos ver informação acerca da entidade escolhida, o período de condução, o tempo do motor em vários estados, a distância percorrida, o consumo médio, o número de excessos efetuados, o número de viagens e serviços, e por fim o *score* do desempenho do condutor.

Figura 2: Ecrã de detalhes de um período de condução

Na Figura 2 pode-se ver o ecrã de detalhes de um dos períodos de condução listados na Figura 1. Este ecrã contém os dados do ecrã principal apresentados de forma mais explícita, com a adição de alguns detalhes, tais como a variação de velocidade, o tipo de excessos de velocidade, a posição no mapa onde os diversos excessos aconteceram, e por fim os micro-periodos que formam este período de condução.

Esta aplicação serviu de base para a UI (User Interface) da aplicação móvel desenvolvida, pois o seu principal objetivo é fornecer estes dados no contexto móvel para ambos os condutores e os gestores poderem visualizar e analisar o desempenho dos mesmos ao longo do tempo, de forma mais ágil.

2.2 DESENVOLVIMENTO MULTI-PLATAFORMA

Alguns dos problemas com impacto no desenvolvimento de *software*, nomeadamente no desenvolvimento móvel, não são tão relacionados com a tecnologia necessariamente, mas sim relacionados com o mercado. O *time to market* tem cada vez ficado mais curto, levando a que seja necessário acelerar o processo de desenvolvimento. Para isto foram criadas *frameworks* multi-plataforma.

Frameworks Multi-Plataforma permitem aos programadores criarem aplicações compatíveis com mais do que um sistema operativo, no caso deste projeto, Android e iOS. Estas *frameworks* possibilitam a escrita de código apenas uma vez, ou seja, com o desenvolvimento de apenas uma base de código, pode-se compilar uma aplicação tanto para Android ou iOS. A grande vantagem desta abordagem é permitir produzir e lançar uma aplicação muito mais rápido para ambos os sistemas, pois ainda que se gaste mais tempo no desenvolvimento da aplicação, poupa-se tempo na adaptação para as várias plataformas.

2.2.1 *Abordagens multi-plataforma mobile*

Existem várias abordagens para o desenvolvimento de aplicações multi-plataforma nomeadamente: abordagem *web*, híbrida, interpretada e *Cross Compiled*. Neste subcapítulo irão ser descritas com mais detalhe estas quatro abordagens de desenvolvimento.

2.2.1.1 *Abordagem Web*

A abordagem *web* é baseada em *web browsers* para dispositivos móveis. As aplicações baseadas nesta abordagem são implementadas com o auxílio de linguagens como HTML (HyperText Markup Language), CSS (Cascading Style Sheets) e JavaScript, as mesmas tecnologias para criar qualquer outro *website*. Estas dependem de um *browser* e do seu *runtime environment* para poderem ser utilizadas. A aplicação é implementada como um *website* otimizado especialmente para dispositivos móveis, sendo assim necessário ter atenção a diferentes tamanhos e resoluções de ecrãs. Uma das vantagens desta abordagem é não haver a necessidade de instalar/atualizar qualquer tipo de componente, pois a aplicação corre no *browser*. Uma desvantagem desta abordagem é que, como é apenas um *website*, este resultará numa pior experiência do utilizador, tanto no que toca no desempenho da aplicação, bem como no que toca à experiência do utilizador, pois as interfaces gráficas destas aplicações são mais limitadas, e podem não seguir o comportamento, ou normas nativas [21] [8] [5] [7]. Um exemplo de uma das tecnologias que se enquadra nesta abordagem é

o JQueryMobile [17], uma biblioteca de javascript especialmente otimizada para interfaces de dispositivos móveis.

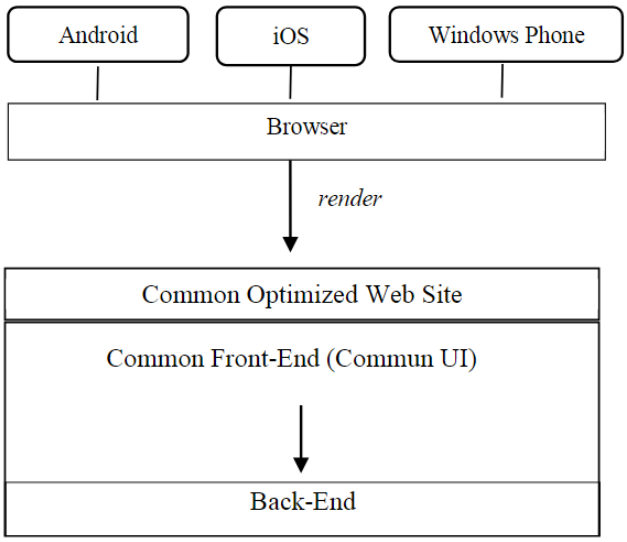


Figura 3: Abordagem Web (adaptado de [21])

2.2.1.2 *Abordagem Híbrida*

A abordagem híbrida é uma combinação entre as tecnologias *web* e as tecnologias nativas. Esta abordagem utiliza o *Browser Engine* do dispositivo e incorpora o conteúdo HTML, CSS, JavaScript numa aplicação nativa (*WebView* no Android, *UIWebView* no iOS) como se fosse uma vista para a página *web*. As funcionalidades nativas de cada plataforma são acessíveis através do uso de uma ponte (*bridge*) que é utilizada na linguagem de programação respetiva de cada *framework*, para a comunicação. Ao contrário da abordagem *web*, as aplicações híbridas podem ser distribuídas pelas lojas de aplicações, no entanto, a interface híbrida tem um desempenho inferior quando comparadas às interfaces nativas, já que a execução é efetuada no *Browser Engine*. Além disso, a *UI* não tem uma aparência nativa, o que se pode tornar num problema [21] [8] [5] [7]. Exemplos de tecnologias que usam esta abordagem são: Apache Cordova [1] e Ionic Framework [13].

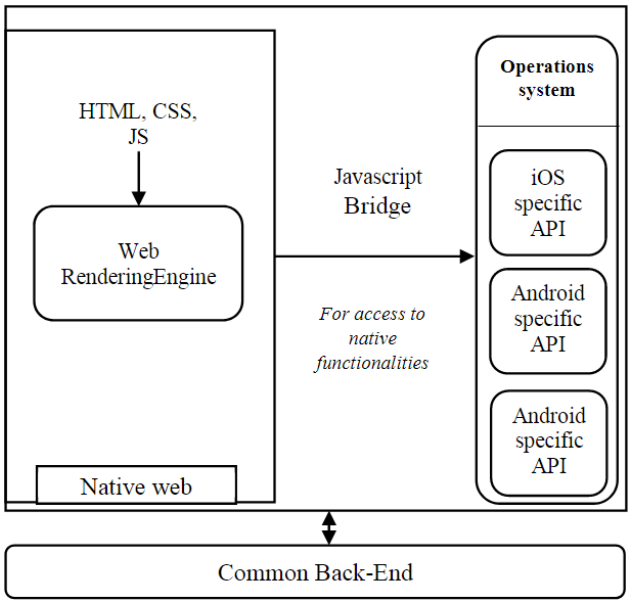


Figura 4: Abordagem Híbrida (adaptado de [21])

2.2.1.3 *Abordagem Interpretada*

A abordagem interpretada utiliza linguagens como o JavaScript para o código relativo à *UI*, e para gerar o código nativo equivalente para cada uma das diversas plataformas, separando o código em várias componentes. As componentes nativas são fornecidas por uma camada abstrata que interpreta o código em *runtime*, para acederem às *API* nativas. A vantagem desta abordagem é que permite a criação de *UI* nativas. Uma das suas desvantagens é a degradação do desempenho das aplicações desenvolvidas, causado pelas constantes chamadas à camada abstrata em *runtime* [21] [8] [5] [7]. Exemplos de tecnologias que usam esta abordagem são: React Native [32] e Titanium [41].

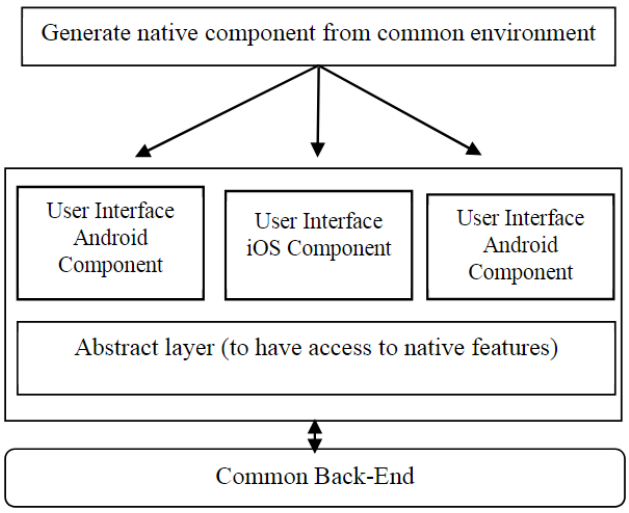


Figura 5: Abordagem Interpretada (adaptado de [21])

2.2.1.4 *Abordagem Cross Compiled*

Nesta abordagem, os programadores escrevem o código utilizando uma linguagem de programação que é futuramente compilada para as diferentes plataformas destino. Os principais benefícios desta abordagem é as aplicações conseguirem atingir quase o desempenho nativo e fornecer todos os recursos e funcionalidades de aplicações nativas, bem como componentes de *UI* nativa. A principal desvantagem é que alguns recursos não são suportados por algumas *frameworks* ao utilizarem esta abordagem [21] [8] [5] [7]. Exemplos de tecnologias que usam esta abordagem são: Xamarin [43] e Flutter [2].

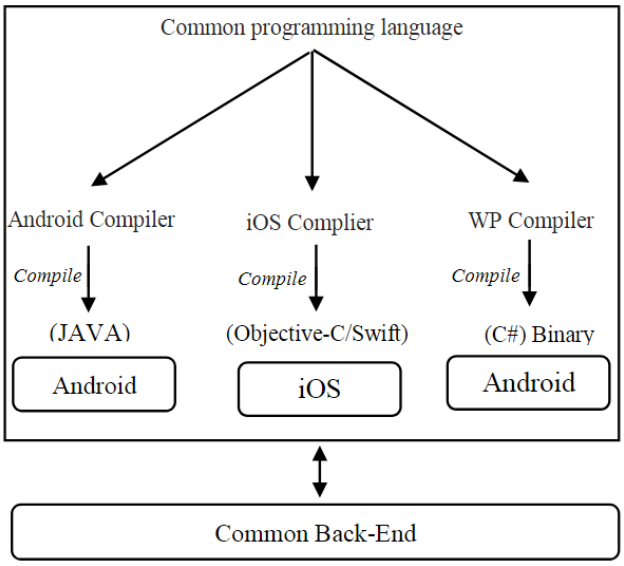


Figura 6: Abordagem Cross Compiled (adaptado de [21])

2.2.2 *Desenvolvimento nativo vs. multi-platform*

O desenvolvimento Nativo de aplicações *vs.* desenvolvimento multi-plataforma é algo que não tem uma resposta certa, pois ambos têm vantagens e desvantagens. O desenvolvimento nativo por norma é mais dispendioso, pois é necessário ter uma equipa para desenvolver Android e iOS independentemente, ou uma equipa que seja capaz de desenvolver para ambas as plataformas, durante um período de tempo mais longo; no entanto, as aplicações têm por norma maior qualidade e melhor desempenho em geral. O desenvolvimento de aplicações multi-plataforma, por outro lado, tem um custo de desenvolvimento mais baixo, pois será apenas necessária uma equipa para desenvolver, o que é uma grande vantagem no mundo empresarial. Por vezes o desempenho destas aplicações pode não ser tão bom como aplicações nativas, bem como a *UI* não ser coerente com a respetiva plataforma, levando a uma

possível pior experiência do utilizador. Na tabela 1 pode-se ver esta comparação ilustrada.

Tabela 1: Comparação entre desenvolvimento nativo e multi-plataforma

	Nativo	Multi-Plataforma
Custo	Custo mais alto	Custo mais baixo
Usabilidade do código	Funciona numa única plataforma	Funciona em várias plataformas e sistemas operativos
Acesso ao dispositivo	Acesso a API nativa com todas as funções	Acesso a API nativas ainda limitado
Performance	Performance Nativa	Boa performance, mas com alguns problemas de compatibilidade

2.3 XAMARIN VS XAMARIN FORMS

Xamarin é uma *framework open-source* da Microsoft para construir aplicações para iOS, Android, MacOS, etc. Esta começou em 2011 com o lançamento do Mono (Android) e MonoTouch (iOS). Xamarin estende a .NET *framework* com bibliotecas específicas para Android, iOS e MacOS.

Com a abordagem tradicional, para criar uma aplicação Android e iOS seria necessário criar 2 projetos completamente independentes, em que, um seria desenvolvido em Swift (iOS) e outro em Java ou Kotlin (Android).

Usando Xamarin Native toda a aplicação é escrita em C#, podendo assim ser aproveitada a camada de negócio, e o *backend* entre as plataformas. No entanto, a UI não é reaproveitada entre plataformas, tendo esta que ser implementada individualmente para cada plataforma, ou seja, se for criada uma UI com um botão, este terá que ser implementado para Android (criar um "Android.Widget.Button") e iOS (criar um UIButton) independentemente. Isto é uma desvantagem porque será necessário repetir bastante código para implementar a UI em iOS e Android de forma independente; no entanto, isto também pode ser uma vantagem, pois a aplicação terá um *feeling* nativo e responsivo, pois o código desenvolvido é posteriormente compilado para as *frameworks* alvo, tendo este uma *performance* nativa.

Xamarin Forms é uma ferramenta construída sobre o Xamarin Native. A grande diferença é que todo o código é reaproveitado, ou seja, tanto a UI como a camada de negócio pode ser reaproveitada, levando à redução de bastante código “repetido” (não é necessariamente repetido, mas sim implementado em iOS e Android, apesar de ser a mesma funcionalidade). Para isto o Xamarin Forms utiliza uma *markup language*, XAML (Extensible Application Markup Language), que abstrai a diferença

entre a implementação da UI em Android e iOS para ajudar o utilizador e acelerar o processo de desenvolvimento. O que isto significa é que ao criarmos, por exemplo, um *Xamarin.Forms.Button*, este será transformado num *UIButton* caso a plataforma seja iOS, ou um *Android.Widget.Button* caso a plataforma seja Android, através dos respetivos SDK (Software Development Kit). Isto é uma grande vantagem, no entanto, se a aplicação que será desenvolvida necessitar de comportamentos específicos por cada plataforma, será mais difícil implementar estas diferenças em Xamarin Forms do que em Xamarin Native. Na Figura 7 podem-se ver ilustradas as diferenças entre o Xamarin Native e o Xamarin Forms.

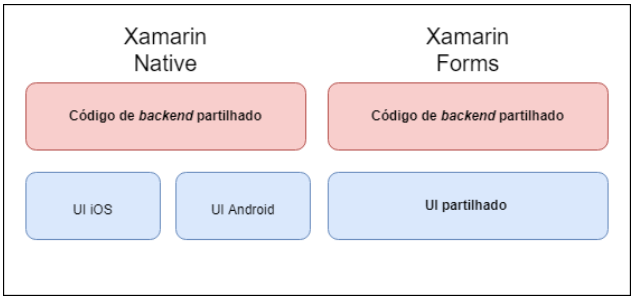


Figura 7: Comparação entre Xamarin Native e Xamarin Forms

Nas tabelas 3 e 2 é feita uma comparação entre Xamarin Native e Xamarin Forms, com as suas Vantagens e Desvantagens.

Tabela 2: Vantagens e Desvantagens do Xamarin Native

Vantagens	Desvantagens
Criar uma UI por plataforma (*feeling* nativo)	Criar uma UI por plataforma
Fácil de ajustar comportamentos específicos por plataforma.	Basicamente está-se a desenvolver código equivalente a Java e Swift, mas em C#
	Bastante código "duplicado"

Tabela 3: Vantagens e Desvantagens do Xamarin Forms

Vantagens	Desvantagens
Criar uma UI para todas as plataformas (Android e iOS)	Desempenho inferior a nativo
90% o código é partilhado	Por vezes pode ser difícil ajustar a UI para diferentes plataformas para comportamentos específicos.
Hot reload desenvolver a UI	

2.4 MVVM

O padrão MVVM (Model-View-ViewModel) é um padrão adaptado por diferentes *frameworks* que organiza o código por *Models*, *Views*, e *ViewModels*, desacoplando assim as várias componentes de uma aplicação. Este padrão também necessita de um mecanismo de *data-binding* para associar uma componente da UI a um *ViewModel*. Como cada componente está desacoplada, realizar testes unitários torna-se muito mais simples porque as unidades de código estão isoladas umas das outras.

O modelo é a componente responsável por definir os dados e lógica de negócio. Este não se preocupa com como é que os dados são mostrados ao utilizador. Esta componente é completamente isolada das outras, ou seja, um modelo bem implementado pode ser futuramente reutilizado por outras aplicações dentro do mesmo domínio, um exemplo disto seria utilizar o mesmo modelo para uma aplicação *web* ou uma aplicação móvel.

A vista é responsável por controlar coisas que interajam diretamente com o utilizador. Esta contém todos os elementos da UI como botões, listas, texto, etc. É aqui que se define uma conexão entre uma componente de uma vista para um *ViewModel* através de *data-binding*. *Data-binding* é um mecanismo que liga dados de uma *View* a um *ViewModel*, por exemplo, ligar uma *ListView* em XAML a uma *List<Object>* em C#.

O *ViewModel* é o intermediário entre a lógica de negócio e as vistas. Os *ViewModels* estão associados a certos componentes da UI, declarados através de data-binding, estes também expõem os modelos para as vistas, para manter registo de dados relevantes para a UI. Um exemplo de uma funcionalidade de um *ViewModel* é o processo de preencher uma lista de itens de certo modelo e associá-lo a uma *View*. Um *ViewModel* também é responsável por aceder a serviços para acesso a dados. Por fim, no *ViewModel* também são tratados os diferentes eventos de uma UI como, por exemplo, os cliques em botões, etc.

2.4.1 *MVVM em Xamarin forms*

Aplicações em Xamarin Forms podem ter várias abordagens, uma delas é usar os *code-behind files*, que estão associados a cada ficheiro XAML, ou seja, para cada ficheiro *{algo}.xaml* existe um *{algo}.xaml.cs*, demonstrado na Figura 8. A maioria do código nestes ficheiros controla o que acontece na UI, sendo exemplos disto a mudança de cor de um elemento da UI, a mudança de texto, gerir eventos de um botão, etc. O problema desta abordagem é a dificuldade de posteriormente se escrever testes unitários para os diferentes componentes.

Para isto utiliza-se o padrão MVVM que resolve este problema ao movermos toda a lógica relacionada com a UI para *ViewModels* que são testáveis através testes unitários. Este padrão por norma apenas é usado por *frameworks* que suportam *data-binding*. Em Xamarin Forms pode-se usar o mecanismo de *data-binding* de cada elemento da UI a um *ViewModel* eliminando assim quase todo o código numa vista.

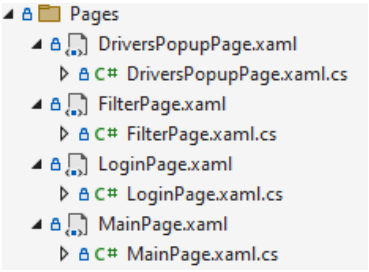


Figura 8: Ficheiros XAML com os ficheiros *code behind* associados.

2.5 ARQUITETURA DE UM SISTEMA

Como já mencionado, a aplicação móvel necessita dois serviços para ser funcional. Um destes serviços é uma API que disponibiliza dados sobre eficiência energética (dados sobre consumo médio de transportes públicos de passageiros, desempenho dos motoristas, excessos cometidos pelos motoristas, etc...). Para a criação desta API teve-se em conta o trabalho já existente no projeto *XTraN Passenger*, visto que este já contém um serviço que fornece estes dados, e que os alimenta à aplicação *XTraN Passenger Web*, apresentada no capítulo **??**; no entanto, não é possível aceder a este serviço vindo de uma fonte externa, tal como uma aplicação móvel. As únicas aplicações que conseguem aceder a este serviço são aplicações contidas no monólito que é o *XTraN Passenger*. Para que este serviço fosse acessível por uma plataforma móvel foi extraído um micro-serviço com as mesmas funcionalidades que o serviço já existe, com o nome de *XtpDriveAPI*. Como O *XTraN Passenger* está a evoluir gradualmente de um monólito para um sistema de micro-serviços, foi feita uma análise destas duas arquiteturas.

2.5.1 *Monólito vs Micro-serviços*

Um monólito é um padrão de desenvolvimento em que todas as componentes da lógica de negócio de uma aplicação estão juntas, ou seja, todas as componentes são distribuídas juntas, todas as componentes são implementadas na mesma tecnologia, todas usam a mesma base de dados, etc. Como estas componentes estão essencialmente todas acopladas umas às outras, isto leva a que o número de dependências

entre as componentes aumente exponencialmente, o que leva a que mudanças no código fiquem mais difíceis com o crescimento de um projeto [26].

Um sistema baseado em micro-serviços consiste em um conjunto de múltiplos componentes (micro-serviços) que comunicam entre si. Por norma, cada micro-serviço implementa funcionalidades de apenas uma componente da lógica de negócio (e.g. um micro-serviço de apenas veículos, outro de linhas, outro de dados de eficiência energética, etc.), sendo estes o mais desacoplados possível, uns dos outros.

Um micro-serviço pode ser, por exemplo, uma RESTful API que é consumida por outros micro-serviços ou por aplicações cliente, ou um sistema de mensagens consumido por outros micro-serviços, ou até o *frontend* de uma página web que consume diversos micro-serviços. É de notar que cada micro-serviço pode ser implementado por tecnologias diferentes, o que pode facilitar o processo de desenvolvimento caso cada equipa esteja apenas responsável pelo seu próprio micro-serviço [6] [26]. Os princípios que devem ser seguidos numa arquitetura de micro-serviços são enumerados a seguir:

- Cada micro-serviço deve ser gerido, replicado, escalado e distribuído independentemente dos outros micro-serviços.
- Cada micro-serviço deve conter apenas uma funcionalidade da lógica de negócio.
- Todos os micro-serviços devem ser o mais *stateless* possível (e.g. RESTful API).
- Deve-se utilizar serviços de confiança para a gestão de estado (e.g. bases de dados, caches, sistemas de ficheiros).

Na Figura 9 pode-se ver a diferença entre uma arquitetura monolítica e uma arquitetura de micro-serviços.

Nas Tabelas 4 e 5 são apresentadas as vantagens e desvantagens de cada uma destas arquiteturas [9].

Tabela 4: Vantagens e Desvantagens uma arquitetura monolítica

Vantagens	Desvantagens
Mais fácil de desenvolver	Fiabilidade (uma falha pode fazer cair toda a aplicação)
Mais simples de distribuir	Disponibilidade (é sempre necessário distribuir toda a aplicação por mais pequena que seja a atualização)
	Mais difícil de escalar a aplicação

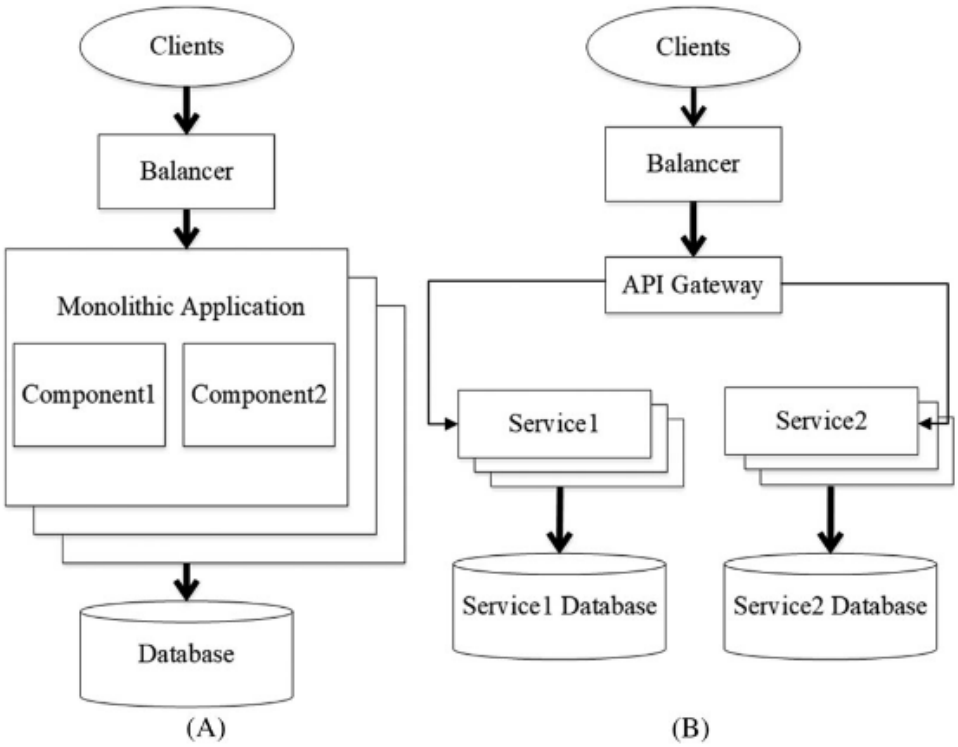


Figura 9: Comparação entre a arquitetura de um monólito (esquerda) vs um sistema de micro-serviços (direita) (adaptado de [44])

Tabela 5: Vantagens e Desvantagens uma arquitetura de micro-serviços

Vantagens	Desvantagens
Manutenção fácil (mais fácil de compreender funcionalidades quando estão divididas por módulos)	Mais difícil de distribuir
Fiabilidade (falha de um micro-serviço só afeta esse micro-serviço)	Mais esforço no processo de desenvolvimento
Disponibilidade (a distribuição de uma nova versão de um micro-serviço requer que apenas esse micro-serviço pare, durante um curto período de tempo)	
Fácil de escalar a aplicação	

A aplicação *XTraN Passenger Web* contém também um processo de autenticação para identificar quem está a consultar informação, e se este tem autorização para fazer esta consulta. Para isto foi criado um micro-serviço de autenticação, onde o principal objetivo deste é fornecer um processo de autenticação, não só para o micro-serviço de eficiência energética utilizar, mas para ser utilizável por todo o *XTraN Passenger* no futuro, bem como todos os micro-serviços extraídos do mesmo. Ambos estes micro-serviços são utilizados pela aplicação móvel desenvolvida neste estágio.

2.6 AUTENTICAÇÃO E AUTORIZAÇÃO

O processo de autenticação consiste em identificar quem é o utilizador. O processo de autorização consiste em identificar ao que o utilizador tem acesso, e o que este pode fazer nos diversos sistemas. No caso de um sistema orientado a micro-serviços este processo não é tão linear, pois ao contrário de uma aplicação implementada como um monólito, todos os micro-serviços estão desacoplados entre si, distribuídos em *containers* diferentes, e possivelmente implementados em diferentes linguagens. Para isto existem várias abordagens que tentam resolver este problema, apresentadas nas secções seguintes.

2.6.1 *Autenticação e autorização em cada micro-serviço*

Uma das abordagens, será ter o processo de autenticação e autorização em cada um dos diferentes serviços [40] [18], como representado na Figura 10. Esta abordagem dá liberdade a que cada micro-serviço tenha a sua própria implementação, o que pode ser considerado uma vantagem, no entanto, há diversas desvantagens:

- A lógica de autenticação e autorização será repetida em cada micro-serviço, o que leva a bastante código repetido.
- Isto faz com que a equipa de desenvolvimento de cada serviço perca o foco de implementar o serviço em si, para implementar um módulo de autenticação.
- É bastante difícil de manter e monitorizar.

Uma opção seria a criação de uma biblioteca que partilhasse código repetido, caso todos os serviços estejam implementados na mesma *framework*; no entanto, esta abordagem continua a não ser escalável, pois caso os diferentes micro-serviços sejam implementados com linguagens diferentes, esta opção acaba por também não ser a melhor.

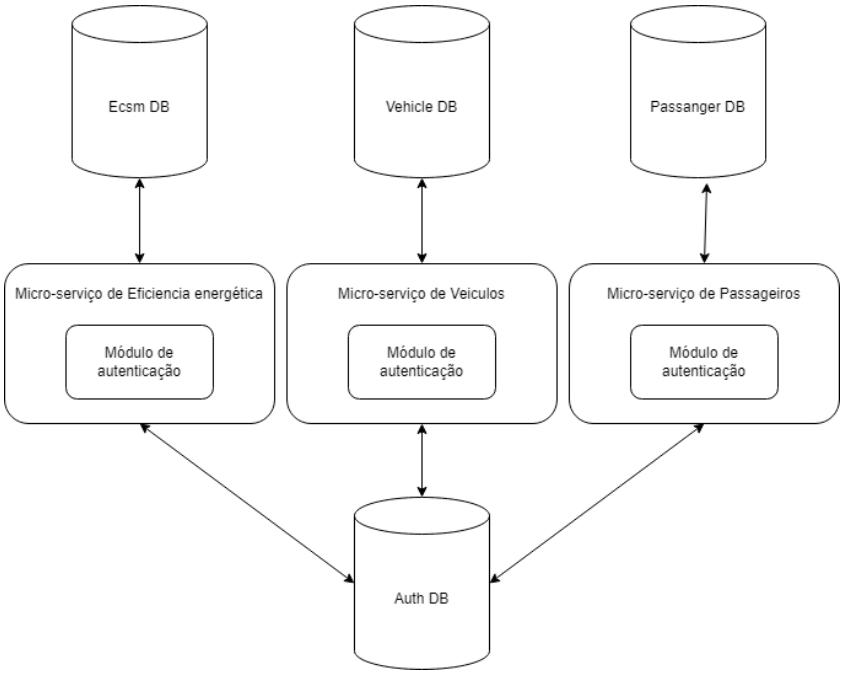


Figura 10: Arquitetura de autenticação e autorização em cada micro-serviço

2.6.2 *Serviço global de Autenticação e Autorização*

Nesta abordagem existe um micro-serviço dedicado a ter responsabilidades de autenticação e autorização, representado na Figura 11. Neste caso, cada micro-serviço irá comunicar com o micro-serviço de autenticação, antes de processar qualquer pedido, para garantir que este está autenticado e que o utilizador tem autorização para cada ação [40] [18]. As desvantagens desta abordagem são as seguintes:

- A velocidade da resposta será influenciada pelo micro-serviço de autenticação, o que aumentará a latência de cada pedido HTTP.
- Dependendo de cada sistema, o processo de autorização é por norma influenciado por regras de negócio, ou seja, este processo não deve ser tratado globalmente.

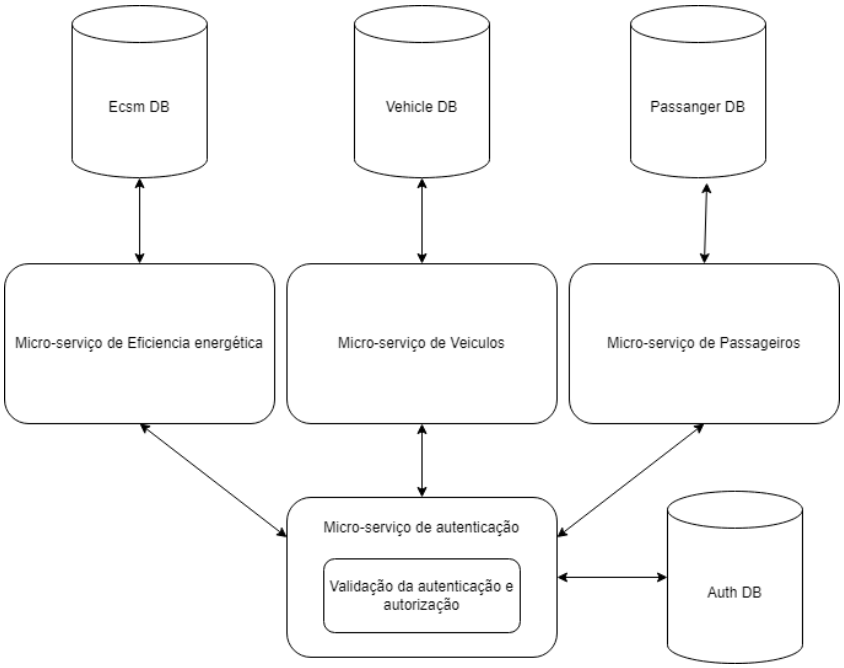


Figura 11: Arquitetura de um serviço global de autenticação e autorização

2.6.3 *API Gateway e Serviço global de Autenticação e autorização por micro-serviço*

Ao mudar para uma arquitetura de micro-serviços, uma das principais perguntas é como os clientes de uma aplicação comunicam com os micro-serviços. Uma abordagem é usar o acesso direto entre cliente e micro-serviço [40] [18]. Esta abordagem sofre de um forte acoplamento entre os clientes e os micro-serviços. Para isto pode ser utilizada uma API gateway que serve como um único ponto de entrada para os serviços, ou seja, o cliente envia pedidos para a API gateway e esta distribui este mesmo pedido para os diversos micro-serviços necessários. Como a API gateway é um único ponto de entrada, este é ideal para fazer o processo de autenticação, como representado na Figura 12. Antes de redirecionar os pedidos para cada micro-serviço, a gateway faz a validação dos parâmetros de autenticação, e a seguir, caso este processo seja bem-sucedido, este gateway irá enriquecer o pedido com dados acerca do utilizador, dados relevantes para autorização como, por exemplo, as *roles* do utilizador. Por fim, quando o pedido chega ao micro-serviço destino, este faz o processo de autorização conforme as diversas regras de negócio necessárias.

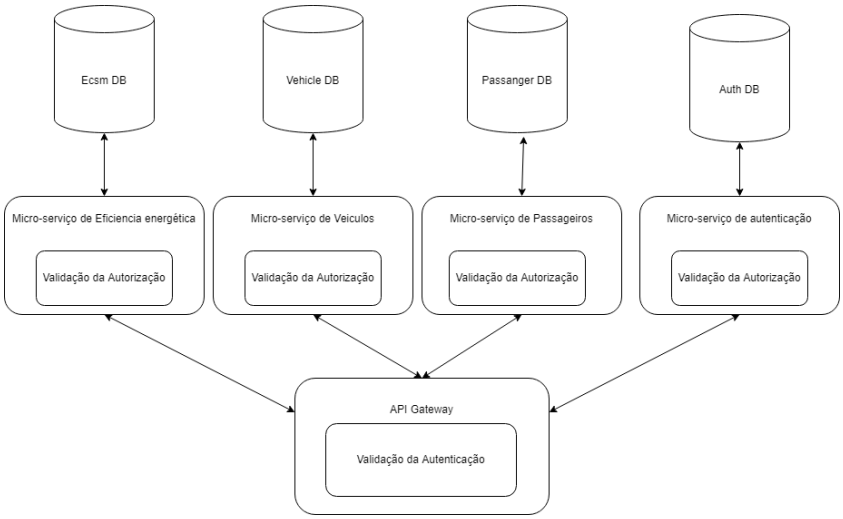


Figura 12: API Gateway e Serviço global de Autenticação e autorização por micro-serviço

2.6.4 *Abordagem utilizada*

Na arquitetura deste projeto decidiu-se utilizar uma API gateway em conjunto com um serviço global de autenticação, e autorização por micro-serviço. Para isto foi criado um serviço de autenticação que gera um JWT (JSON (JavaScript Object Notation) Web Token) no processo de login. Após as aplicações cliente terem um JWT, estas enviam o mesmo nos pedidos HTTP (Hypertext Transfer Protocol) posteriores, que passam pela API gateway. A API gateway valida a assinatura do JWT e encaminha-o para os micro-serviços correspondentes ao pedido feito. Os micro-serviços usam os *claims* e *roles* para fazer o processo de autorização que vêm no JWT.

2.6.4.1 *JWT*

O *JSON Web Token* é um *standard* (RFC (Request for Comments) 7519 [16]) que define uma forma compacta e autónoma de transmitir informação seguramente entre sistemas, através de JSON. Esta informação pode ser verificada e confiada porque é assinada digitalmente. O JWT pode ser assinado usando um *secret* (com o algoritmo HMAC (Hash-Based Message Authentication Code)) ou um par de chaves públicas/privadas (com os algoritmos RSA (Rivest-Shamir-Adleman) ou ECDSA (Elliptic Curve Digital Signature Algorithm)). Quando os tokens são assinados utilizando um par de chaves públicas/privadas, a assinatura certifica que apenas o sistema com a chave privada é que o assinou.

A estrutura de um JWT consistem em três partes: *Header*, *Payload* e *Signature*. O *Header* contém informação acerca do algoritmo utilizado para a assinatura, e o tipo de token, neste caso será do tipo JWT.

O *Payload* contém informação sobre as *claims*. As *claims* são declarações sobre uma entidade (tipicamente, o utilizador) ou dados adicionais que sejam relevantes. Existem três tipos de *claims*: *claims* públicas, privadas e registadas. *Claims* registadas são um conjunto de *claims* predefinidas que não são obrigatórias, mas recomendadas. Algumas delas são: iss (emissor), exp (data de expiração), sub (assunto), aud (audiência), etc. *Claims* públicas são *claims* que podem ser definidas por quem cria o JWT, devendo estas ser registadas no *JSON Web Token Claims Registry*. Visto que este tipo de *claims* não pode interferir com as *claims* registadas, estas são por norma constituídas por um UUID (Universally Unique Identifier) ou por um prefixo de um domínio. *Claims* privadas são *claims* que emissor e consumidor decidem utilizar para a troca de informação, estas devem ser utilizadas com cuidado, pois podem causar conflitos se colidirem com as *claims* registadas.

A *Signature* é a assinatura do JWT, esta é criada com base no *Header*, no *Payload*, e ainda no *secret* ou chave privada do algoritmo utilizado. A assinatura é utilizada para verificar a integridade da mensagem, verificando assim que esta não foi alterada pelo caminho. Caso se utilize um algoritmo assimétrico (com uma chave publica e privada) também se verifica a autenticidade do emissor da mensagem.

O resultado disto é uma *string* composta pelas 3 componentes mencionadas, em base64, separadas por "." . Na Figura 13 pode ser visto um exemplo de um JWT, bem como o seu conteúdo.

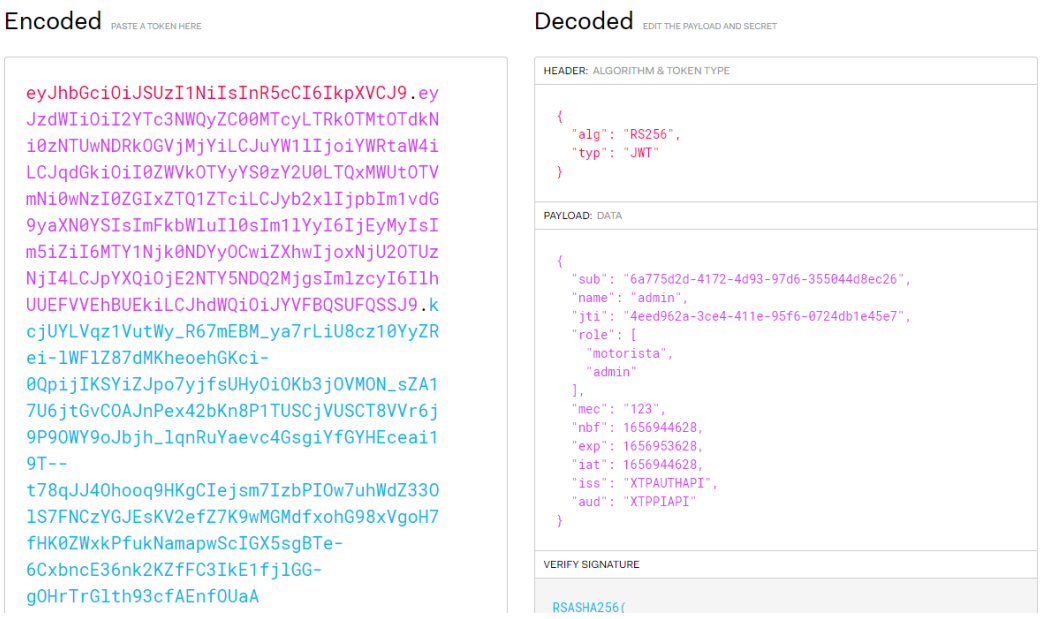


Figura 13: Exemplo de um JWT, bem como o seu conteúdo

2.6.4.2 *API Gateways*

Uma API gateway é um componente que se situa entre aplicações cliente e uma coleção de API. Esta componente age como um *reverse proxy* ao aceitar todos

os pedidos feitos às API, redirecionando os mesmos para os respetivos serviços, e retornando o resultado destes. No entanto, sempre que um pedido é recebido pela API gateway, esta pode executar diversas funções que filtram o pedido. Exemplos destas funções podem ser: restrições por IP (Internet Protocol), *rate limiting* (limite de pedidos feitos num período de tempo), validação do JWT, RBAC (*Role-based Acess Control*).

Para implementar uma API gateway foram investigadas as seguintes tecnologias: Kong [20], Ocelot [27] e Gravitee.io [10]. Cada uma destas tecnologias têm as suas vantagens e desvantagens, por isso foi criado um cenário de teste para cada uma delas para avaliar qual seria mais adequada no contexto empresarial. Na tabela 6 pode-se ver a comparação entre estas tecnologias, focado nas principais funcionalidades que a empresa estava interessada.

Tabela 6: Comparação entre Kong, Gravitee.io e Ocelot

Funcionalidades	Kong	Gravitee.io	Ocelot
GUI (Graphical User Interface)	Sim (em conjunto com outros projetos open-source)	Sim	Não
Restrições por IP	Sim	Sim	Sim
Rate limiting	Sim	Sim	Sim
Validação do JWT	Sim	Sim	Sim
Role-based Acess Control	Sim (com *plugins* open-source)	Sim (difícil de configurar)	Sim
Estatísticas dos micro-serviços	Sim (com *plugins* e customizável)	Sim (não customizável)	Não

O Ocelot foi a primeira tecnologia a ser testada, esta é uma API gateway em .NET, o que no contexto desta empresa seria uma vantagem (pois toda a empresa é baseada em projetos .NET/Microsoft), no entanto, esta foi logo descartada, visto que a empresa valoriza bastante a possibilidade de uma GUI para a configuração do gateway por pessoas que não sejam necessariamente programadores, e o Ocelot apenas permite configurar o gateway através de um ficheiro JSON.

De seguida foram testadas o Kong e o Gravitee em simultâneo. Estas duas tecnologias são bastante idênticas no processo de configuração (através do docker), e ambas contêm essencialmente as mesmas funcionalidades, no entanto, a principal diferença entre estas duas tecnologias, é que o Gravitee.io é mais restrito nas configurações e menos customizável, enquanto o Kong é mais livre e customizável com diferentes *plugins open-source*. Por esta razão, decidiu-se utilizar o Kong como a API *gateway* no projeto.

3

METODOLOGIAS

Metodologias de desenvolvimento de *software* visam planear e estruturar o ciclo de desenvolvimento de *software*. Existem dois principais tipos de metodologias no contexto do desenvolvimento de *software*: metodologias tradicionais e metodologias ágeis.

As metodologias tradicionais de desenvolvimento de *software* são caracterizadas por fases e etapas bem definidas e por uma maior quantidade e qualidade de documentação [39]. Um exemplo deste tipo de metodologia é o *Waterfall*.

As metodologias ágeis de desenvolvimento de *software* são caracterizadas por desenvolvimento iterativo e incremental, para criar diversas versões de protótipos em curtos períodos, e por uma menor quantidade de documentação [22] [39]. Estas metodologias seguem o *Agile Manifesto* [3], sendo os seus principais princípios:

- **Indivíduos e interações** acima de que processos e ferramentas.
- ***Software* funcional** acima de documentação abrangente.
- **Colaboração com o cliente** acima de negociação contratual.
- **Responder à mudança** acima de seguir um plano.

Exemplos de metodologias ágeis são: Scrum [38], Kanban e XP (Extreme Programming) [42].

3.1 METODOLOGIA DE DESENVOLVIMENTO

Neste estágio foi usada uma metodologia de desenvolvimento baseada no Scrum, no entanto, esta não segue à risca todas as etapas do Scrum. A metodologia usada contém sprints semanais onde é feita a retrospetiva da sprint passada, bem como o planeamento de tarefas para a próxima sprint. Para este planeamento é utilizada uma ferramenta interna, com o nome de *Reports*, que permite a criação de tarefas (*tasks*) e relatórios (*reports*), como demonstrado nas figuras 14 e 15.

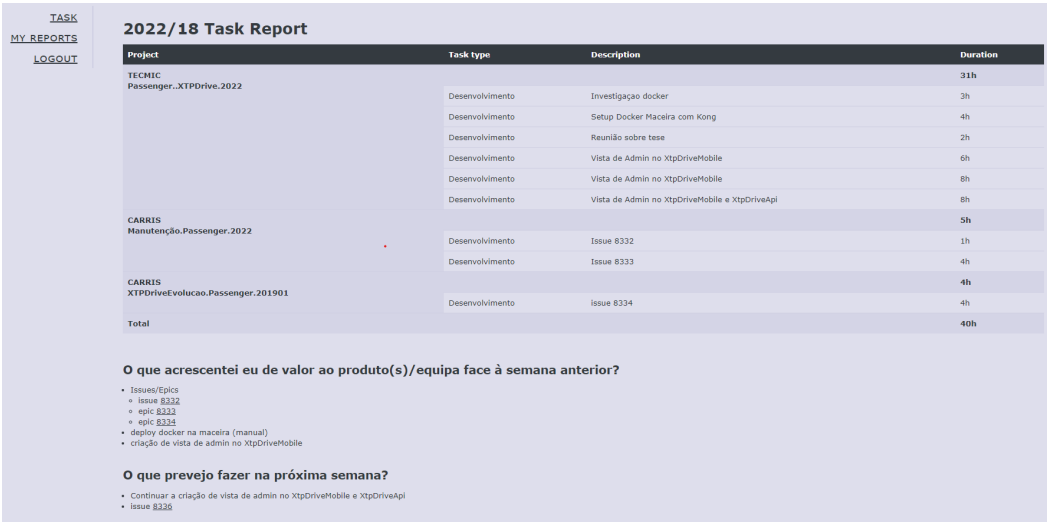


Figura 14: Exemplo de um *report* na ferramenta *Reports*

Figura 15: Exemplo de *tasks* na ferramenta *Reports*

As *tasks* servem para apontar tarefas que foram feitas, enquanto os *reports* são criados semanalmente e contêm as tarefas efetuadas na última semana, criando assim um relatório semanal. Estes *reports* têm também uma zona de planeamento para a próxima semana. Esta ferramenta tem como principal objetivo facilitar a gestão de toda a equipa.

Para auxiliar o desenvolvimento, também é utilizada a ferramenta Azure DevOps TFS (Team Foundation Server) [4]. O Azure DevOps é uma plataforma da Microsoft cujo objetivo é auxiliar o desenvolvimento de *software* em equipa e a gestão dos projetos, através de diversos componentes.

O Azure Boards é o componente que permite criar e gerir quadros de Kanban, bem como a criação dos *Work Items* de *backlog* associados a uma pessoa, estes podem ser: *User Stories*, *Features*, *Epics*, ou *Tasks*.

Cada *Epic* representa uma tarefa geral (e.g. um módulo), e pode conter *Features* filho. Cada *Feature* representa uma tarefe mais específica que um *Epic* (e.g. "implementar autenticação"), e pode conter *User Stories* filho. Cada *User Story* representa

uma tarefa específica que uma *Feature* do ponto de vista do utilizador (e.g "como gestor quero poder ver o dashboard geral"), e pode conter *Tasks* filho. Cada *Task* representa uma tarefa mais específica que uma *User Story* (e.g "implementação do *frontend* em *mobile*").

O Azure Repos é a componente que permite a criação e gestão de repositórios Git. É possível gerir todas as operações de um repositório, tais como: *commits, branches, pull requests*, etc.

O Azure Pipelines é a componente que permite a integração dos repositórios com pipelines de CI/CD (Continuous Integration & Continuous Development). Estes pipelines podem automatizar *builds*, e *deployments*.

O Azure Test Plans é a componente que permite a criação de testes, focando mais em testes de usabilidade, bem como relatórios acerca dos mesmos.

O Azure Artifacts é o componente que permite publicar *packages* públicos ou privados, e facilita o processo de CI/CD dos mesmos. Estes *packages* podem ser nuget, npm, maven, e python *packages*.

No contexto deste estágio foi criado um projeto no Azure DevOps com o nome XtpDrive, representado na Figura 16.

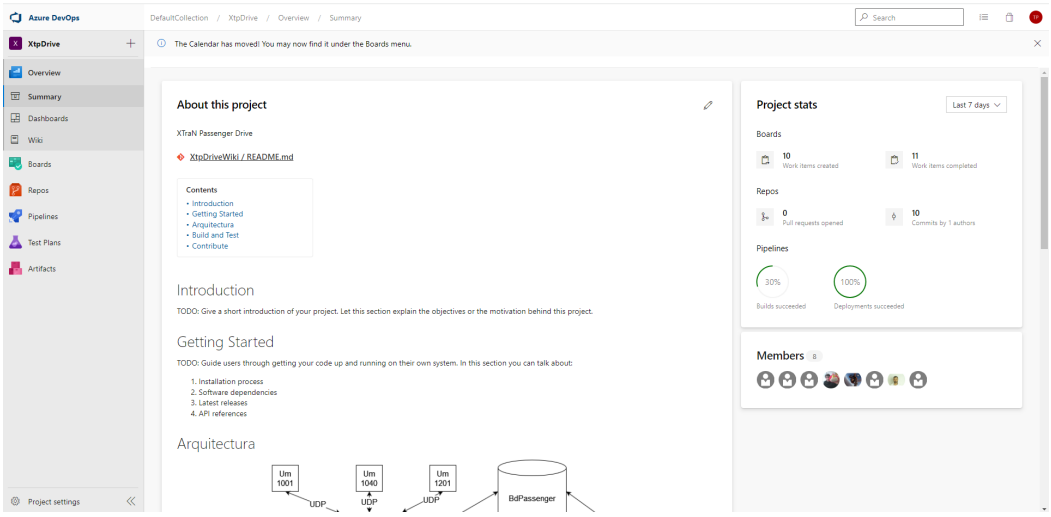


Figura 16: Dashboard do projeto XtpDrive

Neste Projeto foram criados 2 repositórios, um para o *XtpDriveMobile* e outro para o *XtpDriveAPI*. Foi também criado um repositório para o *XtpAuthAPI*, num projeto já existente chamado *XtpPlataforma*, que contém projetos usados por vários outros projetos.

Para cada requisito foi criado um *Work Item* no Azure Boards do respetivo projeto, para identificar tarefas a fazer, sendo estas caracterizadas por um estado, tal como: em desenvolvimento, resolvidas, abertas ou fechadas. Nas figuras 17 a 19 pode-se ver exemplos dos diferentes tipos de *Work Items* utilizados no projeto.

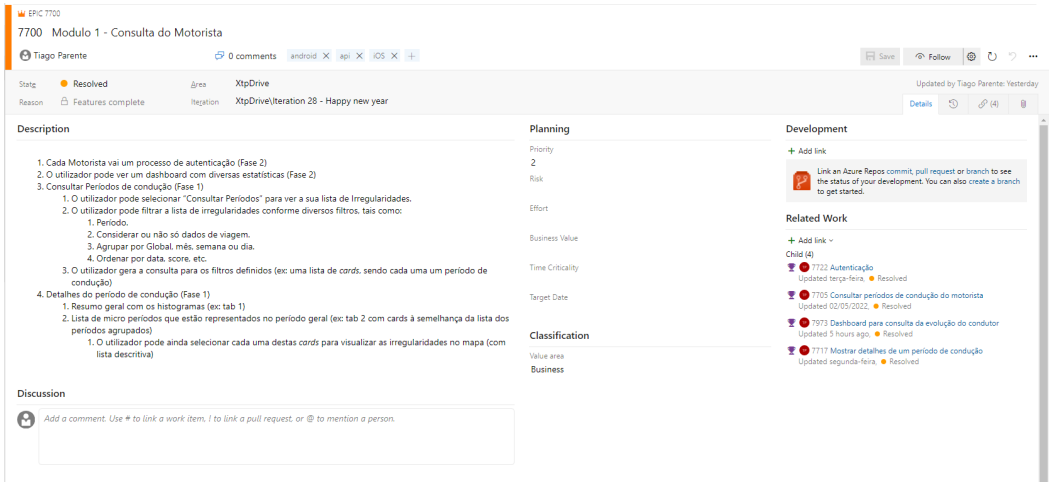


Figura 17: Exemplo de um Epic

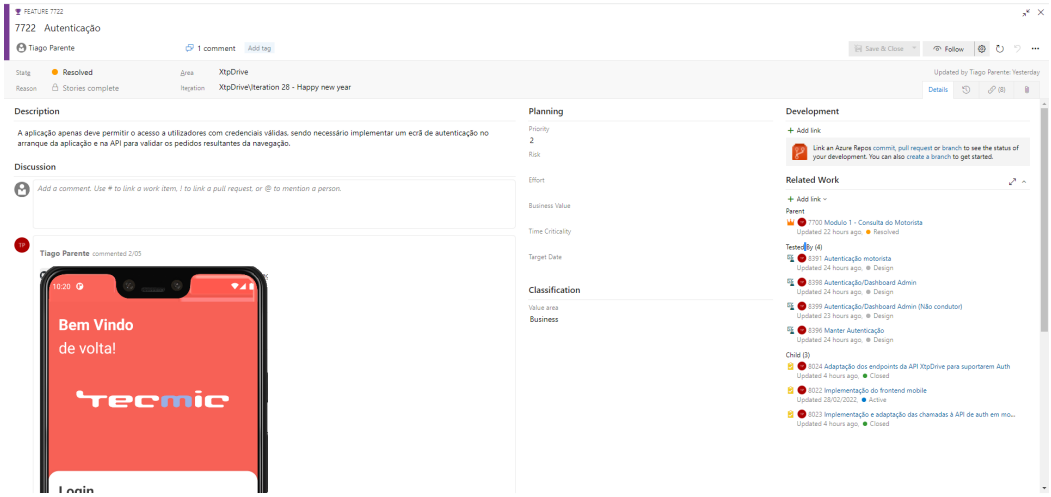


Figura 18: Exemplo de uma feature

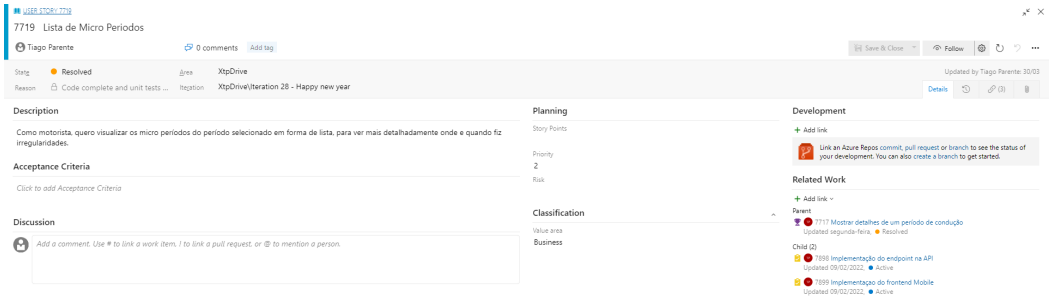


Figura 19: Exemplo de uma User-Story

4

REQUISITOS E ARQUITETURA

A arquitetura desenvolvida neste projeto foi implementada de forma a ser escalável, tendo em conta o sistema atual do *XTraN Passenger*. O *XTraN Passenger* tem como principal objetivo recolher e analisar dados sobre veículos de transporte público, sendo a base onde este projeto foi implementado. Este contem um serviço de eficiência energética que processa dados acerca de consumo dos veículos, desempenho dos motoristas, e excessos cometidos ao longo das viagens feitas.

Como este serviço é necessário para a aplicação móvel ficar funcional, foi extraída uma API que replica o mesmo, fornecendo assim uma forma de aceder aos dados processados pelo *XtraN Passenger*, por aplicações que sejam independentes deste, tal como a aplicação móvel desenvolvida.

Foi também criada uma API de autenticação para a aplicação móvel ter uma forma identificar o utilizador que está a pedir dados, e ao que este tem autorização. Ambas estas API são consumidas pela aplicação móvel criada, sendo necessário que ambas se integrem não só com esta aplicação, mas também com o sistema já existente, sendo assim necessário que a arquitetura seja robusta e escalável.

Este sistema criado tem o nome de *XtpDrive*, sendo esta a razão pela qual a aplicação móvel, apresentada neste capítulo, ter sido renomeada de *XTraN Passenger Bus Monitor Mobile App* para *XtpDriveMobile*.

Neste capítulo é apresentada a arquitetura geral do sistema *XTraN Passenger*, onde este projeto foi integrado. São também apresentados os requisitos funcionais definidos para o projeto, bem como os *mockups* feitos neste contexto. De seguida é apresentada a arquitetura geral com mais detalhe, e por fim, é apresentado o fluxo de autenticação definido neste projeto.

4.1 ARQUITETURA DO SISTEMA XTRAN E CONTEXTUALIZAÇÃO

A arquitetura geral do sistema XTraN Passenger pode ser representada pela Figura 20. Nesta imagem podemos ver a zona que envolve este projeto, sendo esta representada pela cor amarela.

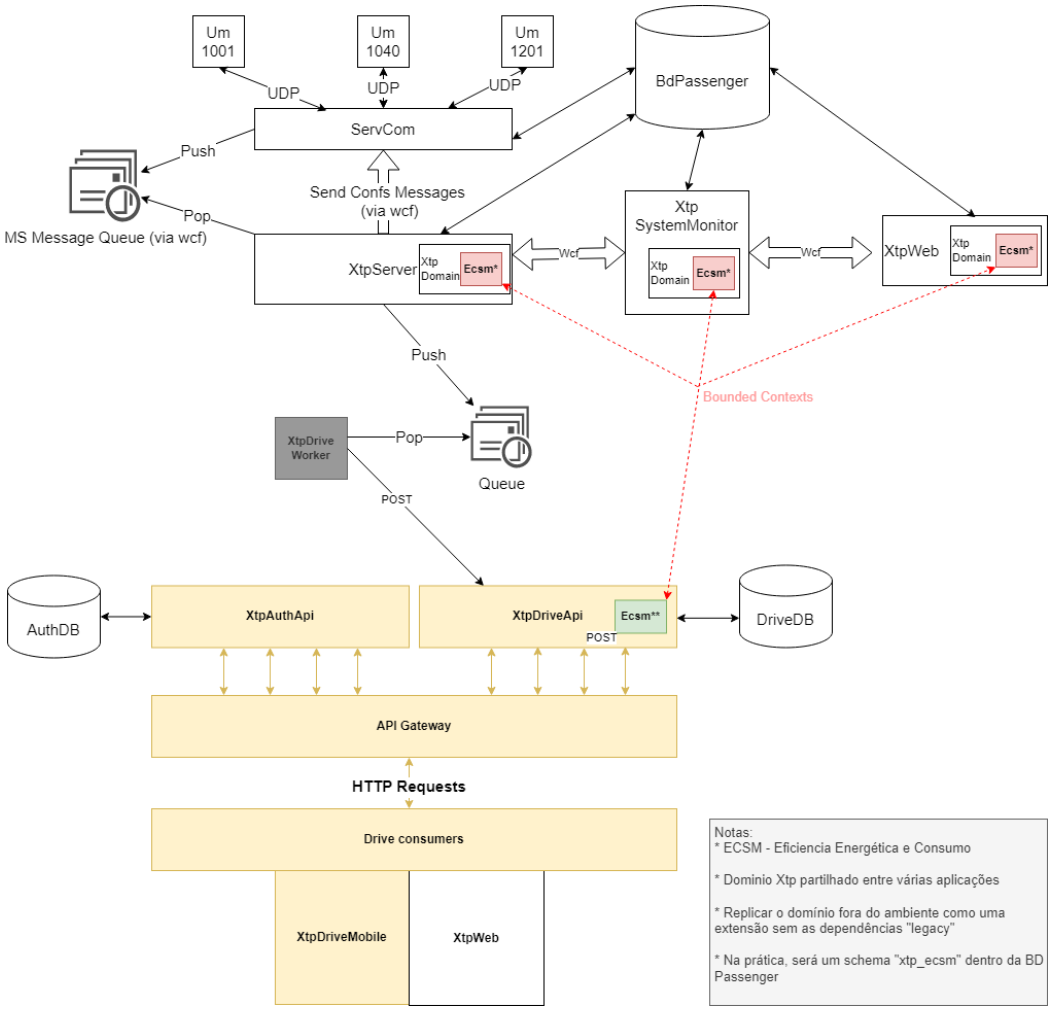


Figura 20: Arquitetura geral do sistema *XTraN Passenger* com a adição deste projeto.

O *XTraN Passenger* é responsável por recolher, analisar e mostrar dados sobre veículos de transporte público, tais como autocarros. Cada autocarro tem uma UM (Unidade Móvel), que é uma mini distribuição Linux, que comunica com o próprio veículo através de interfaces físicas. Cada UM também comunica com o *ServCom* (Servidor de Comunicações), responsável por receber mensagens das UM e responder às mesmas, ou reencaminhar as mensagens para o *XtpServer* ou para a BD (Base de dados) *Passenger*. O *XtpServer* é o responsável processar as mensagens vindas do servidor de comunicações, e por vezes responder às UM. Este é também responsável por comunicar com a BD *Passenger*, bem como comunicar com outros diversos serviços. Um destes serviços é o *XtpSystemMonitor*, responsável por permitir que o *XtpWeb* receba atualizações em tempo real vindos do *XtpServer*. Também é por aqui que o *XtpWeb* pode interagir com as UM, sendo estas mensagens processadas e redirecionadas para o *XtpServer*.

É neste contexto que se integra o trabalho descrito nesta tese. O principal objetivo foi criar uma aplicação móvel de *back office* para que os condutores de transportes

publico de passageiro possam consultar o seu perfil, onde estão representados todos os seus dados do módulo de eficiência energética. Exemplos destes dados são:

- o número de serviços;
- o número de viagens efetuadas;
- o consumo por cada serviço/viagem que estes efetuam;
- o número de excessos que estes efetuam;
- o número de cada tipo de excesso como, por exemplo, excesso de aceleração, travagem, rotação, etc;
- um número de 0 – 10 que representa o seu *score*;

Para isto ser possível é necessário isolar o modelo de domínio responsável pelos dados de eficiência energética, ao criar um micro-serviço para este módulo. A principal razão para isto é o monólito que é o *XTraN Passenger* estar a ficar com uma dimensão cada vez maior, e com o número de dependências a aumentar, alterações no código começam a ser cada vez mais difíceis de implementar, visto que é necessário sempre verificar que todo o sistema continua funcional após qualquer mudança. Com um serviço isolado de eficiência energética, apenas é necessário verificar que este funciona isoladamente sem problemas, e que os outros serviços/aplicações comunicam com ele sem problemas, evitando assim dependências entre projetos.

Neste contexto foi então decidido que seria criada uma *RESTful* API responsável apenas por dados de eficiência energética, com o nome de *XtpDriveAPI*. Esta deveria depois ser partilhada pelo projeto *Web* já existente, bem como a aplicação móvel desenvolvida neste estágio. Foi também decidido que seria criada uma API de autenticação, com o nome de *XtpAuthAPI*, responsável por identificar o utilizador que está a fazer pedidos HTTP, e ao que este utilizador têm acesso.

4.2 REQUISITOS FUNCIONAIS

Os requisitos funcionais e *mockups* (apresentados na secção 4.3) definidos para a aplicação móvel, foram definidos em articulação com o orientador na empresa. Estes foram definidos ao início do processo de desenvolvimento, e foram feitas alterações (maioritariamente apenas na UI) iterativamente, com base no feedback dos protótipos criados. De seguida são apresentados estes requisitos:

1. (Módulo 1) Cada motorista terá um processo de autenticação;
2. O utilizador pode ver um dashboard com diversas estatísticas do módulo ECSM;
3. Consultar períodos de condução;

a) O utilizador pode selecionar um botão “Consultar Períodos” para ver a sua lista de irregularidades/excessos;

b) O utilizador pode filtrar a lista de irregularidades/excessos conforme diversos filtros, tais como:

i. Intervalo de tempo;

ii. Considerar ou não só dados de viagem;

iii. Agrupar por mês, semana ou dia;

iv. Ordenar por data, *score*, etc;

c) O utilizador pode gerar uma consulta para os filtros definidos, sendo apresentada como resultado uma lista de elementos que representam um período de condução;

4. O utilizador pode selecionar um período de condução para ver os seus detalhes, que serão apresentados em 2 *tabs*:

a) *Tab* 1 - Resumo geral com os histogramas;

b) *Tab* 2 - Lista de micro períodos que formam o período selecionado, representados à semelhança da lista dos períodos de condução;

i. O utilizador pode ainda selecionar cada uma destes elementos para visualizar as irregularidades/excessos no mapa (com lista descritiva)

5. (Módulo 2) Cada administrador terá um processo de autenticação;

a) O administrador vai poder escolher o condutor que pretende observar.

b) Caso o administrador também seja condutor, este poderá observar tanto as suas estatísticas, como também as estatísticas dos outros condutores, que este escolher.

4.3 MOCKUPS DA APLICAÇÃO MÓVEL

Os *mockups* realizados para a aplicação móvel estão representados na Figura 21. Estes *mockups* têm em conta a aplicação *web XTraN Passenger Web*, representada nas Figuras 1 e 2.

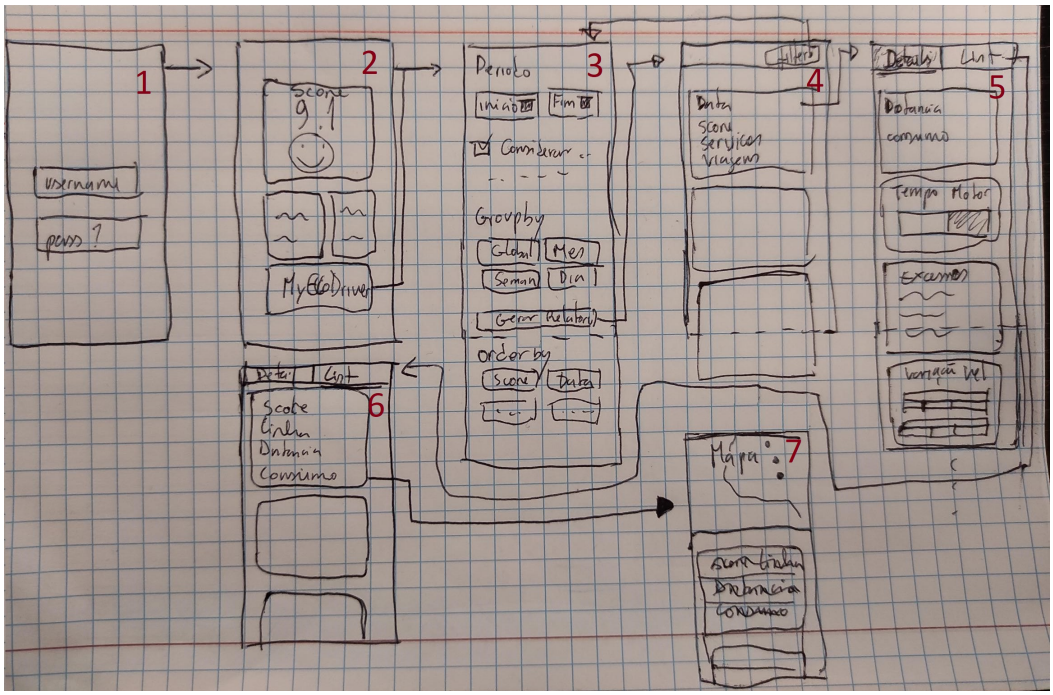


Figura 21: *Mockups* da aplicação móvel.

O fluxo da aplicação começa por um ecrã de Login (1), onde os condutores vão inserir o seu *username* e *password* para entrar. A seguir é apresentado um Dashboard (2) onde são mostrados dados, gráficos e estatísticas do utilizador nos últimos 6 meses. O dashboard contém também um botão "*MyEcoDriver*" que irá abrir um ecrã de filtros (3) para o utilizador escolher os dados que quer consultar. No fim de escolher os diversos filtros, é apresentada ao utilizador uma lista de períodos de condução (4), com as suas estatísticas. Cada elemento desta lista é clicável, navegando assim para detalhes deste período de condução (5). O ecrã de detalhes (5) contém duas *tabs*, uma de detalhes gerais e outra de micro períodos. A *tab* de detalhes gerais (5) contém estatísticas, dados e gráficos correspondentes ao período selecionado. A *tab* de micro períodos (6) contém uma lista dos diversos micro períodos que constituem o período selecionado. Cada um destes micro períodos é também clicável, navegando para um ecrã de detalhes dos mesmos. Este ecrã de detalhes de micro períodos (7) é constituído por um mapa e uma lista descritiva, que contém os diversos locais onde o condutor realizou irregularidades/excessos.

4.4 ARQUITETURA GERAL

Na Figura 22 está representada a arquitetura geral do sistema *XtpDrive* mais detalhadamente, apresentando todos os componentes desenvolvidos neste projeto em conjunto, e como estes interagem entre si.

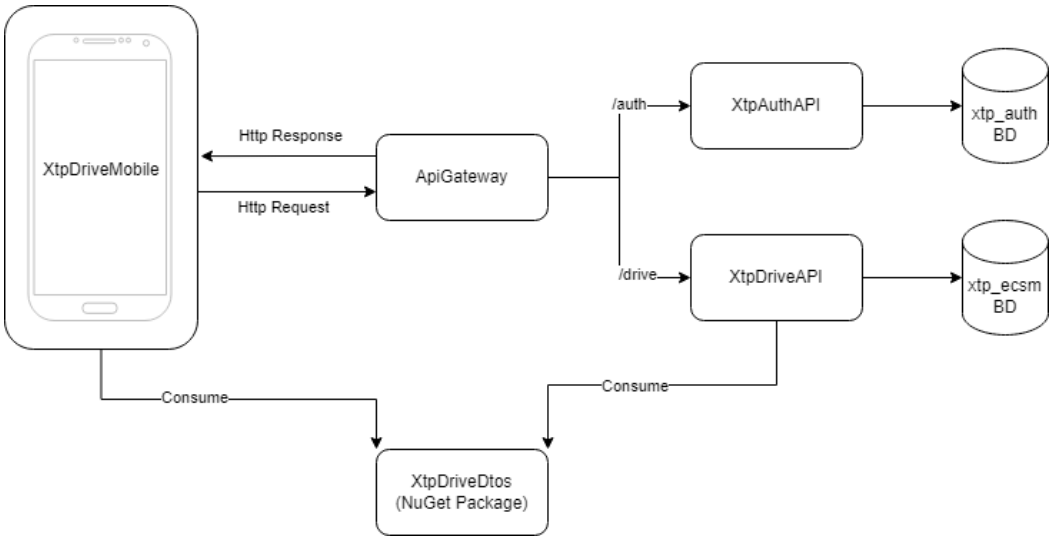


Figura 22: Arquitetura do *XtpDrive*.

Como já mencionado, esta aplicação móvel visa apresentar, aos condutores de transporte público, dados sobre a eficiência energética na sua condução. Para isto é utilizada a *XtpDriveAPI* responsável por dados de eficiência energética e a *XtpAuthAPI* responsável pelo processo de autenticação de utilizadores. No entanto, para aceder a ambas estas API foi decidido também utilizar uma API gateway que valida a autenticação em todos os pedidos HTTP e restringe os mesmos se necessário. É também partilhado entre a *XtpDriveAPI* e o *XtpDriveMobile* um NuGet Package que contêm todas as classes DTO (Data Transfer Object), para facilitar a partilha destas entre os projetos.

A *XtpAuthAPI* é responsável pelo processo de *login*, que essencialmente passa por gerar um JWT caso o utilizador forneça credenciais validas, e posteriormente popular esse JWT com os roles e permissões que o utilizador tem. Este JWT será futuramente incluído no cabeçalho de todos os pedidos HTTP feitos tanto à *XtpDriveAPI* como à *XtpAuthAPI*, para validar que o utilizador está autenticado, e para estas API decidirem ao que o utilizador tem acesso.

A *XtpDriveAPI* é responsável por dados de eficiência energética dos motoristas. Como todos os pedidos HTTP têm um modelo de dados partilhado pela aplicação móvel e pela *XtpDriveAPI*, foi criado um NuGet Package que contém todas as classes de DTO necessárias, evitando assim a duplicação de código entre aplicações que consumam esta API.

Estes DTO são utilizados para passar dados da aplicação móvel para a API. Estes são passados no corpo ou no cabeçalho do pedido HTTP para, por exemplo, indicar um filtro nos dados ou simplesmente para serializar e desserializar dados das respostas HTTP.

Uma das principais componentes desta arquitetura é a API Gateway, responsável por encaminhar os pedidos HTTP realizados para os diversos micro-serviços, bem como fazer um processo de validação do JWT para verificar que o utilizador está de facto autenticado. Neste caso todos os pedidos para a *XtpDriveAPI* devem ser autenticados, enquanto a *XtpAuthAPI* é aberta, não sendo necessária a validação do JWT.

4.5 FLUXO DE AUTENTICAÇÃO DA *XTPAUTHAPI*

Para gerir *roles* e *claims* existem várias abordagens. Como já mencionado na subsecção 2.6.4 decidiu-se utilizar uma abordagem que inclui uma API gateway em conjunto com um serviço de autenticação (*XtpAuthAPI*), e cada micro-serviço gere a sua autorização.

Sendo assim, os utilizadores do sistema são guardados numa BD do *XtpAuthAPI*. Isto cria um problema que é conseguir associar dados de cada micro-serviço a um utilizador da *XtpAuthAPI*. Para resolver este problema decidiu-se que é necessário que cada *User* seja identificado por um campo único para cada micro-serviço, o que leva a que seja possível mapear dados dos respetivos micro-serviços para um utilizador específico.

Tendo em conta apenas a *XtpDriveAPI*, como todo o tipo de dados estão associados a um *Driver*, e um *Driver* pode ser identificado pelo campo *MecNumber*, concluiu-se que cada *User* na *XtpAuthAPI* terá de conter também um campo *MecNumber*, sendo assim possível realizar o mapeamento de todos os dados da *XtpDriveAPI* para um utilizador da *XtpAuthAPI*. É de notar que com a criação da *XtpAuthAPI*, a tabela de *Drivers* na *XtpDriveAPI* deixa de ser necessária, pois os dados entre *Users* e *Drivers* são repetidos, no entanto, para fazer esta alteração é necessária a alteração do código legado do *XtpServer* que regista os dados.

Cada *User* é também identificado por *N roles*. Estes *roles* identificam a que micro-serviços o *User* tem acesso. Por exemplo, um *User* com os *roles* de *Administrator* ou *Driver* vão ter acesso à *XtpDriveAPI*, no entanto, um *User* com o role de *Finance* já não terá acesso à *XtpDriveAPI*, mas sim a um micro-serviço relacionado com finanças. Para validar este tipo de autorização, a API gateway valida em cada pedido HTTP os *roles* que vêm no JWT, e decide se deve encaminhar ou não o mesmo. Este processo tem como nome de RBAC.

Em cada micro-serviço decidiu-se que é necessária uma tabela de *claims* que identificam as permissões que cada utilizador tem nesse mesmo micro-serviço, através de certas regras de negócio. Por norma cada *claim* está associada a um *role*, ou seja, *Users* que tenham a *role* de *Administrator* vão ter associados as *claims* de *can-read*

e *can-write*, enquanto *Users* que tenham a *role* de *Driver* apenas vão ter associado a *claim* de *can-read*. Assim sendo, em cada pedido feito à *XtpDriveAPI*, é inicialmente validada a assinatura do JWT, a seguir é feito o mapeamento das roles para *claims*, e por fim se o utilizador for autorizado, este segue para a ação do controlador.

Na Figura 23 pode-se ver a amarelo o fluxo do processo autenticação, e a vermelho o fluxo de um pedido autenticado a um micro-serviço. A *XtpAuthAPI* contém um par de chaves pública e privada. A chave privada é utilizada para assinar o JWT gerado. A chave pública é utilizada para validar a assinatura do JWT em *endpoints* que necessitem de autenticação. Para além disto, esta chave pública é também utilizada pela API Gateway com o intuito de validar os pedidos que necessitam de autenticação, bem como para a possibilidade de fazer o processo de RBAC para cada micro-serviço. É de notar que para esta abordagem funcionar, todos os micro-serviços e API gateways que utilizem o *XtpAuthAPI* como serviço de autenticação vão necessitar de partilhar a mesma chave pública.

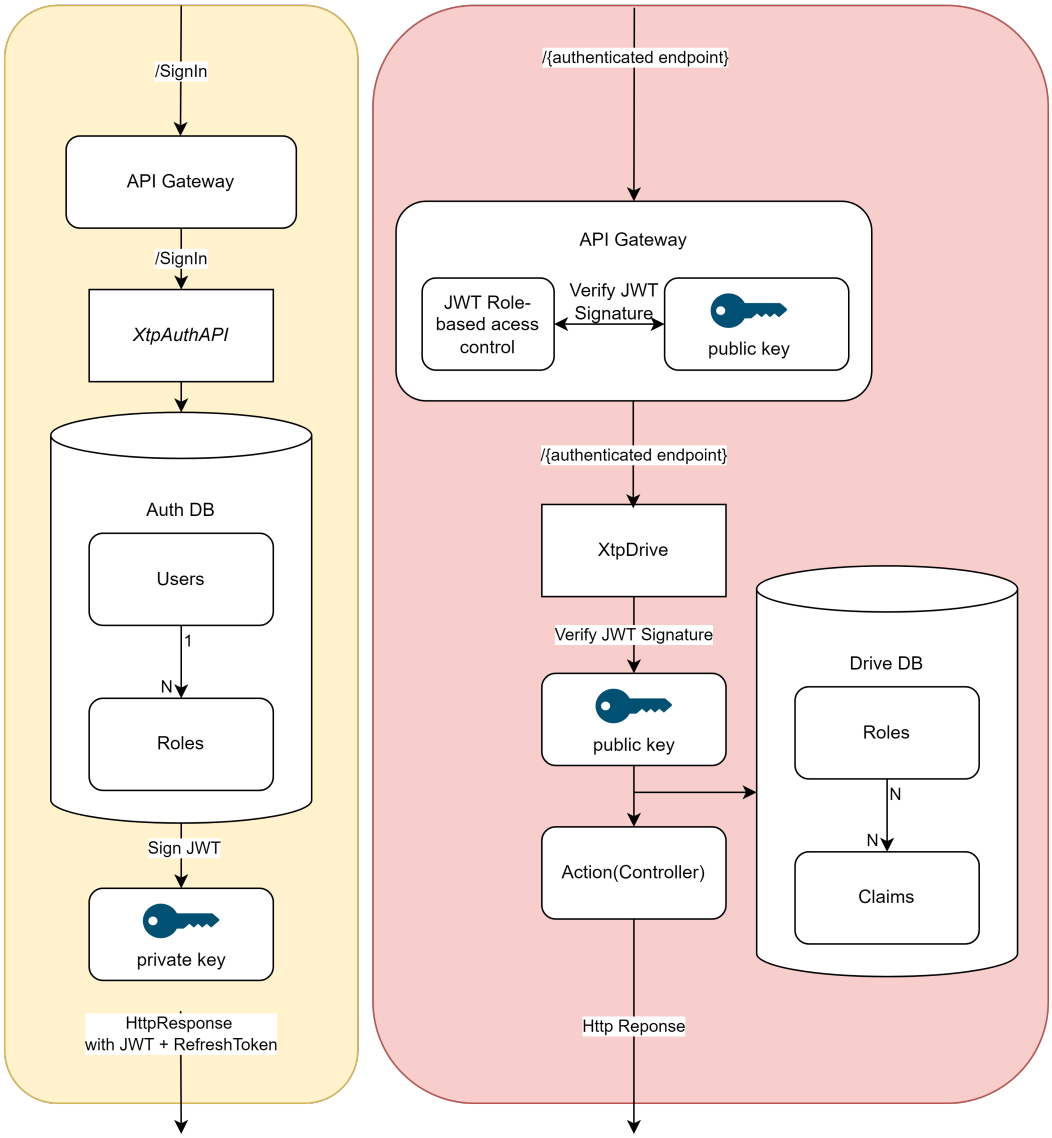


Figura 23: Fluxo do processo de autenticação.

5

TRABALHO DESENVOLVIDO: DESCRIÇÃO DA APLICAÇÃO MÓVEL XTPDRIVEMOBILE E DAS API QUE A SUPORTAM

Neste estágio foram desenvolvidas várias componentes necessárias para implementar o *XtpDriveMobile* (conhecido inicialmente por *XTraN Passenger BUS Monitor Mobile App*). *O XtpDriveMobile* é uma aplicação móvel que representará informação sobre dados de eficiência energética acerca de condutores de autocarros, sendo estes condutores os clientes finais. Para isto a aplicação visa demonstrar o *score* do condutor para a empresa onde este trabalha, bem como a sua evolução semanal. Com esta aplicação o condutor pode também visualizar que tipo de excessos fez, para os poder corrigir no futuro.

No entanto, para ser possível desenvolver esta aplicação, foram necessárias outras componentes, sendo estas: a *XtpDriveAPI*, que gere os dados de eficiência energética; a *XtpAuthAPI* que gere o processo de autenticação; uma API Gateway que valida a autenticação dos *endpoints* e aplica algumas limitações às API se necessário; e o *XtpDriveDtos* que é um NuGet Package para partilha de classes DTO. Estas componentes vão ser descritas mais detalhadamente nas subsecções seguintes.

Na secção 5.1 é apresentada a *XtpAuthAPI*, na secção 5.2 é apresentada a *XtpDriveAPI*, na secção 5.3 é apresentado o NuGet Package *XtpDriveDtos*, na secção 5.4 é apresentada a API gateway utilizada neste projeto, e por fim na secção 5.5 é apresentada a aplicação *XtpDriveMobile*.

5.1 XTPAUTHAPI

A *XtpAuthAPI* é uma RESTful API, cujo objetivo é criar um mecanismo de autenticação que seja futuramente utilizado por todos os serviços do *XTraN Passenger*. Atualmente, mesmo depois do desenvolvimento dos micro-serviços, o sistema *XTraN Passenger* continua um monólito, pois este ainda não foi adaptado para utilizar estes micro-serviços. Sendo assim, este serviço de autenticação ainda é apenas utilizado pela *XtpDriveAPI*, apresentada na secção 5.2, e pelo *XtpDriveMobile*, apresentado na secção 5.5.

Esta API visa também gerir *roles* e utilizadores, ou seja, foram implementados *endpoints* para a gestão de *roles*, bem como a sua relação a utilizadores.

Esta API foi desenvolvida em .NET5, seguindo o padrão de uma Minimal API [34]. Foram também utilizadas diversas bibliotecas/packages para o auxílio do desenvolvimento.

Uma destas bibliotecas é a *Entitiy Framework Core*, um ORM (Object–relational mapping) que facilita e agiliza o processo de acesso a bases de dados usando objetos .NET, eliminando a maioria do código da camada de acesso a dados. Para a utilização desta biblioteca é necessária a criação de classes para o modelo de domínio, bem como a configuração das mesmas na classe *DbContext*.

Outra biblioteca importante nesta API foi a *ASP.NET Identity* [33], uma biblioteca para gerir dados relacionados com autenticação, tais como utilizadores, passwords, *roles*, *claims*, etc. Esta biblioteca já nos fornece os modelos de dados necessários para a gestão de todos os cenários relacionados com autenticação, sendo possível adicionar campos e tabelas se necessário.

Por fim, a última biblioteca utilizada nesta API foi a *ASP.NET Authentication* [24], uma biblioteca facilita o processo de identificar ao que um utilizador tem acesso, através de um *middleware* de autenticação. Ao registar este *middielware* é possível indicar o esquema de autenticação que desejamos, como, por exemplo, JWT *Bearer Tokens*, *Basic Authentication*, etc. É também possível indicar em cada *endpoint* ou controlador se estes devem ser autenticados ou não, através do atributo *Authorize*. Este atributo pode também limitar o *endpoint* por *policies* ou *roles* específicos.

5.1.1 *Modelo de domínio*

O modelo de domínio deste serviço tem em conta as entidades criadas pelo *ASP.NET Identity*, apresentado na Figura 24. Este é assim composto pelas classes *User*, *Role* e *RefreshToken*.

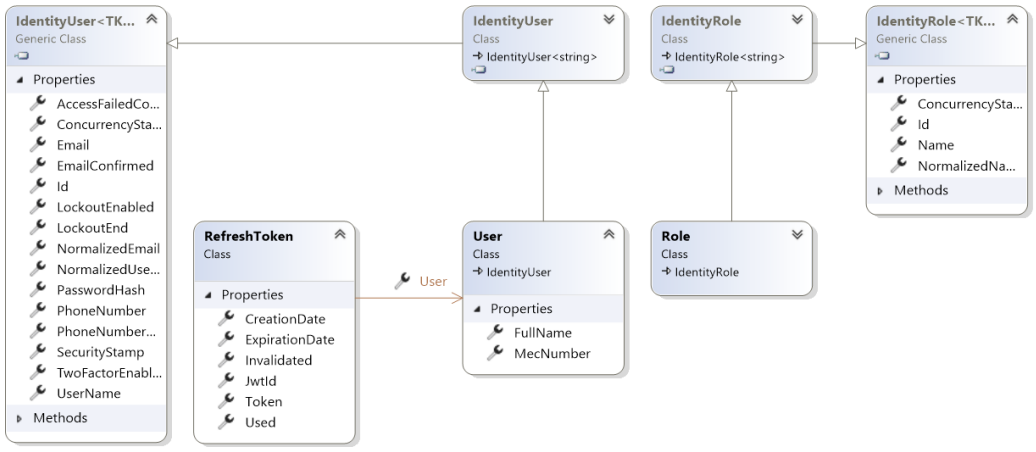


Figura 24: Diagrama de classes do modelo de domínio da *XtpAuthAPI*

A classe *User* é o modelo que representa um utilizador, esta estende a classe *IdentityUser* que vem da biblioteca *ASP.NET Identity*. A classe *Role* é o modelo que representa o papel de um utilizador, esta estende a classe *IdentityRole* que também vem da biblioteca *ASP.NET Identity*. A classe *RefreshToken* é o modelo que representa um token, cujo objetivo é renovar a sessão de um utilizador. Na Figura 25 pode-se ver a estrutura da base dados resultantes destas classes.

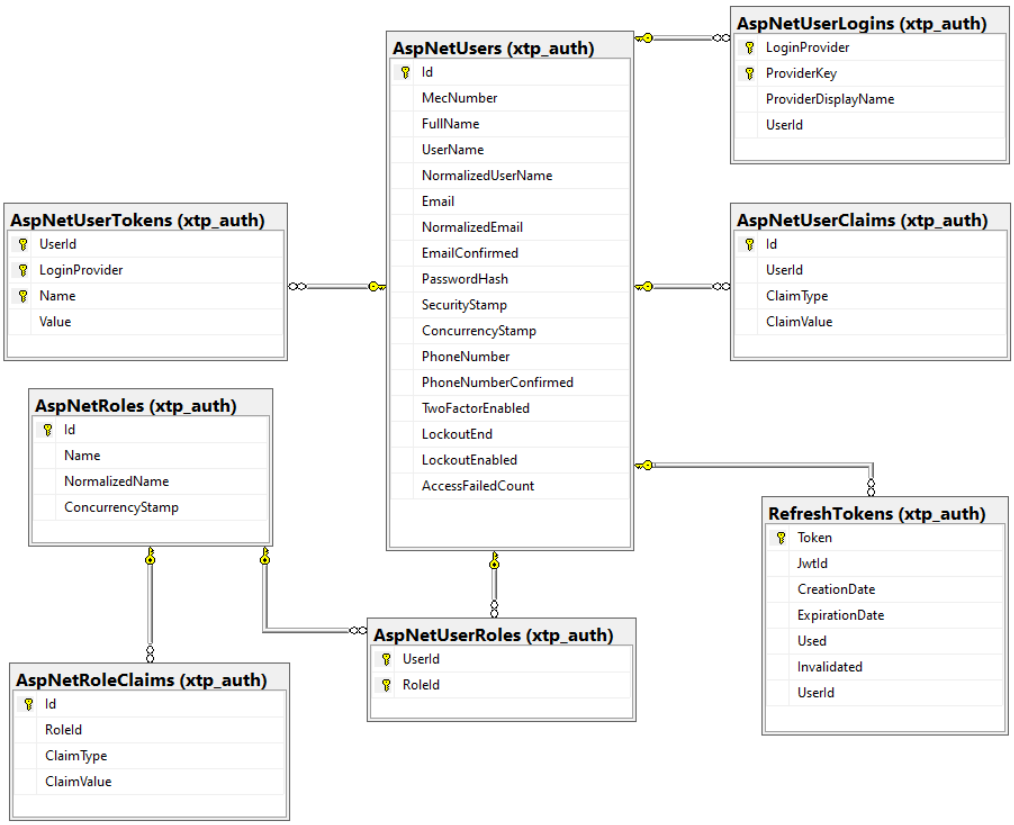


Figura 25: Diagrama da BD da *XtpAuthAPI*

5.1.2 *Roles*

A gestão de roles é algo necessário no contexto do *XTraN Passenger* para identificar o que cada utilizador tem acesso. Esta gestão apenas pode ser feita por utilizadores com role de *Admin*. Para isto foram criados os seguintes *endpoints*:

- GET /roles (apenas para *admins*)
- POST /roles (apenas para *admins*)
- DELETE /roles (apenas para *admins*)

Estes *endpoints* utilizam a classe *RoleManager*, vinda da biblioteca *ASP.NET Identity*, para facilitar o processo de gestão de roles, tal como criar um role, eliminar um role, e obter um role da BD.

5.1.3 *Utilizadores e Autenticação*

A gestão de utilizadores e autenticação consiste na criação/registo de utilizadores, e no processo de login. O login implementado é um processo de autenticação *stateless*, feito através de JWT. É de notar que no contexto do sistema *XTraN Passenger*, apenas faz sentido um administrador efetuar o processo de criação/registo de um utilizador. Para isto foram criados os seguintes *endpoints*:

- POST /signup (apenas para *admins*)
- POST /signin
- POST /refresh

5.1.3.1 *Sign Up*

O processo de registo de um utilizador é feito neste *endpoint*, com o auxílio da classe *UserManager*, vinda da biblioteca *ASP.NET Identity*, para a criação do utilizador. Para isto o controlador recebe um modelo de dados constituído por um *Username*, *Password* e *MecNumber* (opcional).

5.1.3.2 *Sign In*

O processo de *sign in* é também feito com o auxílio da classe *UserManager* para obter o utilizador por *username*, e para verificar se a password recebida corresponde à do utilizador. Assim que isto é verificado, é gerado um JWT.

Para o processo de geração de um JWT foi criado um par de chaves pública e privada através da ferramenta *OpenSSL* [28]. Este par de chaves é utilizado pelo algoritmo de assinatura do JWT, neste caso, o algoritmo assimétrico utilizado foi o *RsaSha256*. A chave privada tem com principal função assinar o JWT no processo de criação do mesmo, e a chave pública tem como principal função validar a assinatura de um JWT. Neste caso esta chave pública deve ser utilizada por todos os outros micro-serviços que utilizem este serviço de autenticação para conseguirem validar a autenticidade do JWT. No contexto deste projeto, ambas as API *XtpAuthAPI* e *XtpDriveAPI*, e a API gateway utilizam esta chave pública.

Decidiu-se utilizar um algoritmo assimétrico para reforçar a segurança do sistema, sendo assim impossível forjar um JWT válido, sem acesso à chave privada, que neste caso é apenas utilizada pela *XtpAuthAPI*.

Quando é criado o JWT são atribuídas ao mesmo diversos *claims* e uma data de expiração. Estes *claims* contêm as seguintes informações:

- *sub*: contém o ID do utilizador.

- *name*: contém o *username* do utilizador.
- *jti*: contém o ID do JWT.
- *mec*: contém o número mecanográfico do utilizador.
- *role*: contém os roles do utilizador.

Na Figura 13 pode-se ver um exemplo de um JWT gerado.

Quando é gerado o JWT é também gerado um *RefreshToken* associado ao mesmo que é guardado na base de dados. Este *RefreshToken* pode ser utilizado futuramente no *endpoint /refresh* para evitar que o utilizador tenha de efetuar o processo de login constantemente.

A resposta HTTP deste pedido contém o JWT gerado, o *RefreshToken* gerado, a data de expiração do JWT e dados acerca do utilizador, tal como o *username*, o *MecNumber*, e os *roles* do utilizador. A principal razão pela qual os dados do utilizador são retornados no pedido HTTP de */signin* é para facilitar as aplicações cliente a obter esta informação, evitando que estas tenham de descodificar o JWT para obter os dados.

5.1.3.3 *Refresh*

O *endpoint /refresh* consiste num *endpoint* que recebe um JWT e um *RefreshToken* para renovar a sessão do utilizador ao criar um novo JWT e um novo *RefreshToken*, sendo assim possível que o utilizador mantenha sessão do ponto de vista de uma aplicação cliente, sem ter que voltar a fazer o processo de login. Por norma, um *RefreshToken* tem uma data de validade maior que um JWT, por uma questão de segurança. Assumindo que os JWT têm uma data de validade de 2 horas, a cada 2 horas será necessário que seja feito o *refresh* do mesmo. Desta forma as aplicações clientes podem verificar se o JWT está expirado, e caso assim seja, é feito o pedido de */refresh*.

Este método verifica que o JWT recebido é válido, que o *RefreshToken* é válido, que este não está expirado, e que este pertence ao utilizador dono do JWT dado. Assim que esta validação é feita, são criados um novo JWT e um novo *RefreshToken*. Ambos estes novos *tokens* são retornados na resposta HTTP, bem como dados acerca do utilizador, tal como no *endpoint /signin*.

5.2 XTPDRIVEAPI

A *XtpDriveAPI* é uma *RESTful API* desenvolvida em .NET5 para devolver dados de eficiência energética. Foram utilizadas diversas bibliotecas para o auxílio ao desenvolvimento desta API.

Tal como na *XtpAuthAPI*, nesta API também foi utilizada a biblioteca *Entity Framework Core*, para facilitar as operações efetuadas numa BD. Foi também utilizada a biblioteca *ASP.NET Authentication* para validação do JWT, bem como a extração de *roles* e *claims* do mesmo, da mesma forma que foi implementado na *XtpAuthAPI*.

5.2.1 *Modelo de domínio*

O modelo de domínio desta API é baseado no modelo de domínio já existente no *XTraN Passenger* (ver arquitetura 4), já que o principal objetivo ao criar esta API é desacoplar tudo o que seja relacionado com o módulo de eficiência energética para um novo micro-serviço. Todas as classes desta API podem ser observadas na Figura 26

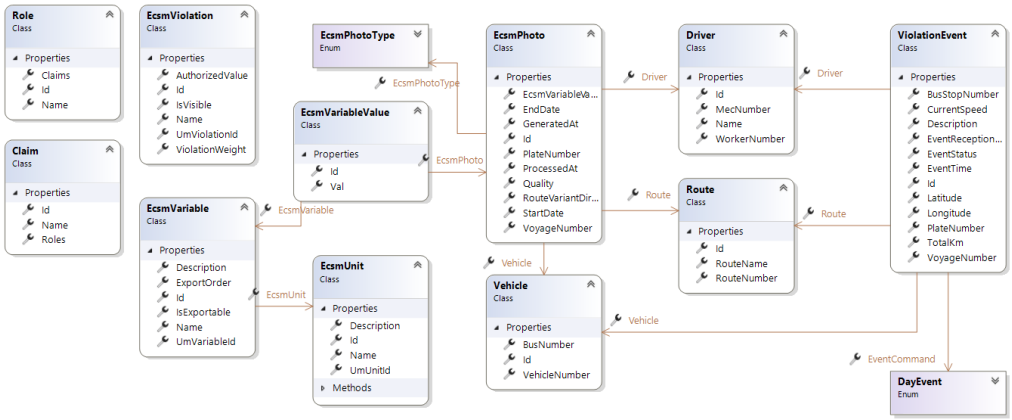


Figura 26: Diagrama de classes do modelo de domínio da XtpDriveAPI

A classe *Driver* é o modelo que representa um condutor. A classe *Route* é o modelo que representa um itinerário. A classe *Vehicle* é o modelo que representa um veículo. A classe *EcsmPhoto* é o modelo que representa um micro período de condução, ou seja, representa os dados estatísticos de eficiência energética num intervalo de tempo. A classe *EcsmUnit* é o modelo que representa uma unidade no contexto do módulo de eficiência energética; por exemplo, uma unidade pode representar metros, segundos, etc. A classe *EcsmVariable* é o modelo que representa uma variável no contexto do módulo de eficiência energética; por exemplo, esta

pode ser distância percorrida, número de viagens, etc. A classe *EcsmVariableValue* é o modelo que representa o valor de uma *EcsmVariable*.A classe *EcsmViolation* é o modelo que representa um tipo de excesso no contexto do módulo de eficiência energética. A classe *ViolationEvent* é o modelo que representa o evento de quando um excesso é cometido por um condutor.

Estas classes são mapeadas para a BD através do *Entity Framework Core*. O diagrama que resulta deste mapeamento pode ser visto na Figura 27.

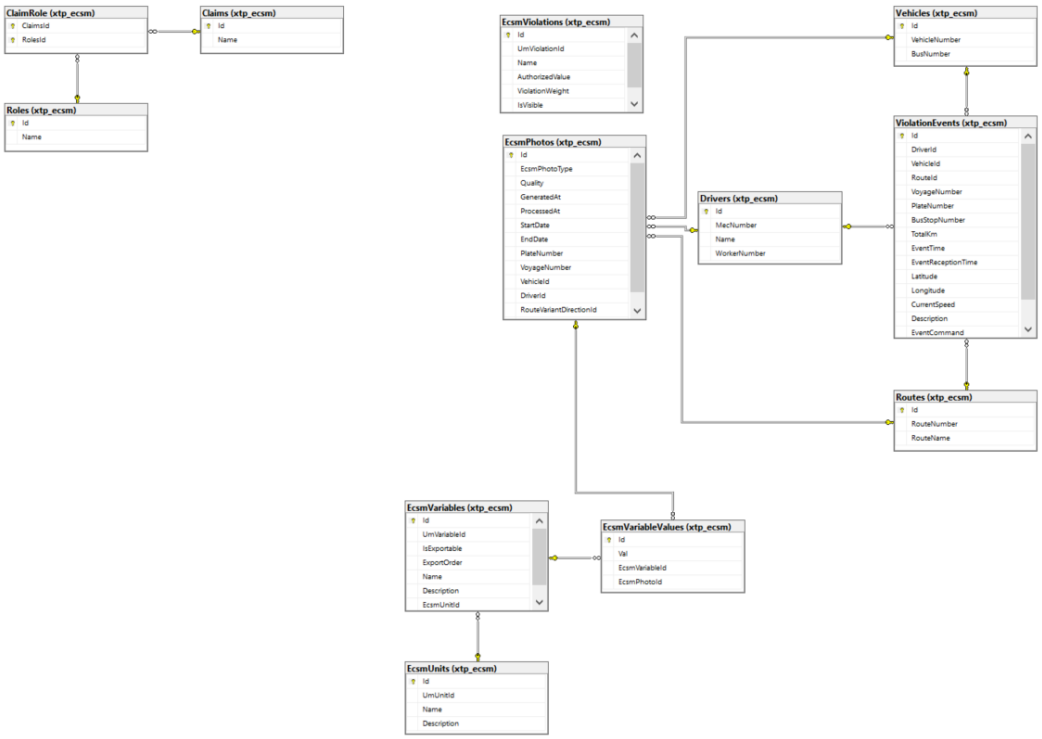


Figura 27: Diagrama da BD drive

5.2.2 *Controladores e Endpoints*

Todos os *endpoints* desta API são de leitura de dados, ou seja, métodos HTTP do tipo GET. É de notar que no futuro serão implementados também os *endpoints* necessários para a escrita e processamento destes dados, mas atualmente este processo é feito pelo *XtpServer*. Esta API está organizada por controladores que vão ser apresentados a seguir.

5.2.2.1 *DriverEcsmPhotos*

Este controlador retorna dados acerca eficiência energética de um utilizador. Como já mencionado, uma *EcsmPhoto* representa um micro-período de condução. Cada um destes micro-períodos tem associado ao mesmo uma data de início e fim. Por

vezes estas *EcsmPhotos* podem ter duração de minutos ou até horas. No entanto, do ponto de vista de uma aplicação cliente esta informação é maioritariamente relevante se estas *EcsmPhotos* forem agrupadas por dia, semana ou mês, para o utilizador final poder observar a sua evolução e *score* diário, semanal ou mensal. Para isto, os modelos retornados nas respostas HTTP maioritariamente agrupamentos de dados, sendo estes definidos no projeto *XtpDriveDtos*, apresentado na secção 5.3, para serem partilhados com o *XtpDriveMobile*, apresentado na secção 5.5.

Os filtros deste controlador, em todos os *endpoints*, são passados por *query* no URL (Uniform Resource Locator), sendo o modelo deste filtro um *EcsmFilterDto*, apresentado na secção 5.3

Todos os *endpoints* deste controlador recebem o *MecNumber* como parâmetro do *endpoint*, pelo que estes têm uma regra de negócio de autorização em que apenas podem ser acedidos por utilizadores que tenham o *MecNumber* indicado no parâmetro, ou por utilizadores com a role *Admin*. Os *endpoints* deste controlador são enumerados de seguida:

- GET /{MecNumber} (apenas *users* com *MecNumber* ou *Admin*)
- GET /{MecNumber}/Result (apenas *users* com *MecNumber* ou *Admin*)
- GET /{MecNumber}/Result/Photos (apenas *users* com *MecNumber* ou *Admin*)
- GET /{MecNumber}/Result/Photos/{photoId}/Violations (apenas *users* com *MecNumber* ou *Admin*)

O *endpoint* GET */DriverEcsmResults/{MecNumber}* tem como função retornar dados das *EcsmPhotos* do utilizador com o *MecNumber* indicado, agrupadas por um período definido nos filtros deste *endpoint*. O modelo devolvido por este *endpoint* é uma *PaginatedResponse<EcsmResultDto>*, apresentado na secção 5.3.

O *endpoint* GET */DriverEcsmResults/{MecNumber}/Result* tem como função retornar detalhes de um *EcsmResultDto*. Para isto, é necessário indicar o *MecNumber* do utilizador, bem como as datas de início e fim do *EcsmResultDto* no filtro do *endpoint*. O modelo devolvido por este *endpoint* é um *EcsmResultDetailsDto*, apresentado na secção 5.3.

O *endpoint* GET */DriverEcsmResults/{MecNumber}/Result/Photos* tem como função retornar uma lista de *EcsmPhotos* que constituem um *EcsmResultDto*. Para isto, é necessário indicar o *MecNumber* do utilizador, bem como as datas de início e fim do *EcsmResultDto* no filtro do *endpoint*. O modelo devolvido por este *endpoint* é uma *PaginatedResponse<EcsmResultPhotoDto>*, apresentado na secção 5.3.

O *endpoint* GET */DriverEcsmResults/{MecNumber}/Result/{photoId}/Violations* tem como função retornar uma lista de *EcsmViolationEventDto* que constituem uma

EcsmPhoto. Para isto, é necessário indicar o *MecNumber* do utilizador e o ID da *EcsmPhoto* da qual o utilizador quiser adquirir dados dos excessos. O modelo devolvido por este *endpoint* é um *IEnumerable<EcsmViolationEventDto>*, apresentado na secção 5.3.

5.2.2.2 *Drivers*

Este controlador tem como principal objetivo retornar dados acerca de condutores. Atualmente este controlador é constituído por apenas um *endpoint /drivers* que retorna a lista de condutores que existem. O modelo devolvido por este *endpoint* é um *IEnumerable<DriverDto>*, apresentado na secção 5.3.

5.2.2.3 *RolesClaims*

Este controlador tem como função a gestão de *Roles* e *Claims* do serviço. Como já mencionado, foi definido que cada micro-serviço irá ter *Claims* do próprio serviço para facilitar o desenvolvimento de regras de negócio no processo de autorização. Para isto existem *Roles* que são uma cópia das roles do *XtpAuthAPI*. Existem também *Claims* em que cada *Claim* está associada a um *Role*, ou seja, cada *Role* tem *N Claims*, e cada *Claim* pode pertencer a *N Roles*.

Este controlador tem os seguintes endpoints:

- GET /roles (apenas para *Admins*)
- POST /roles (apenas para *Admins*)
- DELETE /roles (apenas para *Admins*)
- GET /claims (apenas para *Admins*)
- POST /claims (apenas para *Admins*)
- DELETE /claims (apenas para *Admins*)
- POST /roles/{role}/claims/{claim} (apenas para *Admins*)

5.3 XTPDRIVEDTOS

Este projeto visa facilitar a partilha de classes DTO, que vão ser utilizadas nos pedidos e respostas HTTP, entre a *XtpDriveApi* e o *XtpDriveMobile*. Para isto este projeto é disponibilizado através de um *NuGet Package*. Um *NuGet Package* é uma biblioteca em .NET. que pode ser consumido por outros projetos facilmente. Desta forma, é evitada a duplicação das classes DTO em todos os projetos que consumam a *XtpDriveAPI*, sendo assim possível apenas importar esta biblioteca e utilizar os

modelos necessários. Na Figura 28 pode-se ver estas classes representadas num diagrama.

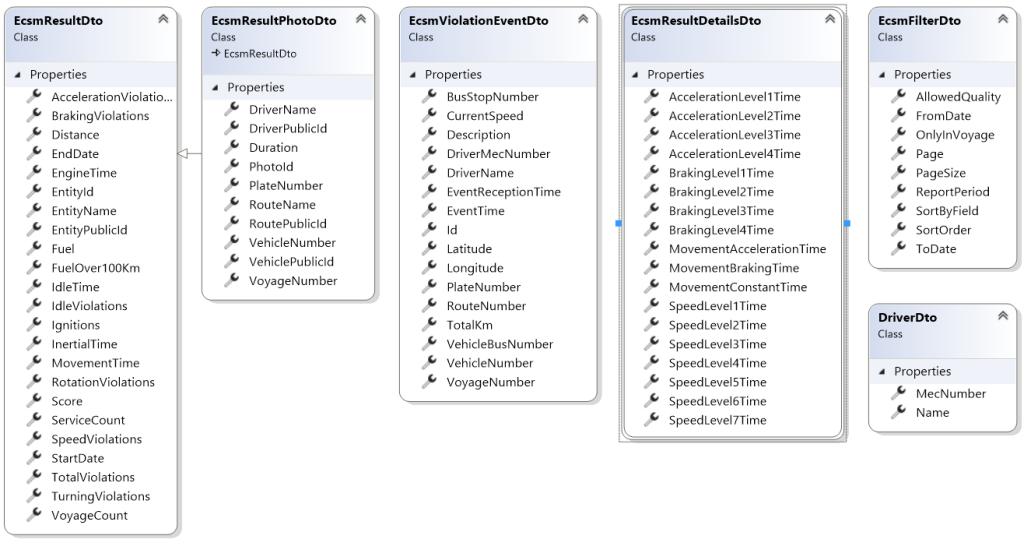


Figura 28: Diagrama de classes do *XtpDriveDtos*

Uma *PaginatedResponse* representa uma resposta paginada que contém dados do tipo T, sendo T qualquer tipo de objeto. Um *EcsmResultDto* é um modelo um agrupamento de *EcsmPhotos*. Um *EcsmResultDetailsDto* é um modelo que representa dados mais detalhados acerca de um agrupamento de *EcsmPhotos*. Um *EcsmResultPhotoDto* é um modelo representa dados detalhados de apenas uma *EcsmPhoto*. Um *EcsmViolationEventDto* é o modelo que representa dados de um excesso. Um *DriverDto* é o modelo que representa dados de um *Driver*. Um *EcsmFilterDto* é o modelo que representa os filtros que o utilizador pode passar por *query* no URL do controlador *DriverEcsmResults* da *XtpDriveAPI*.

5.4 API GATEWAY

A API Gateway aplicada neste projeto foi feita através da tecnologia Kong [20]. Para isto foi composto um ficheiro *docker-compose*, bem como um Dockerfile que contém toda a configuração necessária para que a API gateway seja facilmente configurada. Este gateway funciona como um *reverse proxy* que aceita os pedidos HTTP dos micro-serviços configurados, e encaminha-os os mesmos. Neste caso o gateway também efetua o processo de validação da autenticação nos diferentes micro-serviços. No entanto, também há a possibilidade de limitar e restringir certos acessos aos micro-serviços, se necessário, por exemplo, é possível aplicar regras de *rate-limiting* (limitar acessos num intervalo de tempo) ou até restringir acesso de um IP específico, entre outras funcionalidades. É também possível fazer um processo de RBAC se necessário.

Para instalar o Kong é necessário escolher entre uma abordagem de *containers*, ou um sistema operativo com base em Linux, por exemplo, o Ubuntu. Visto que a empresa está a tentar evoluir para soluções de micro-serviços, e esta apenas tem máquinas Windows Server, foi então decidido utilizar a abordagem de *containers*, através do Docker, para ser possível fazer a distribuição do Kong numa máquina Windows Server, a correr um *container* baseado em Linux.

5.4.1 *Docker Compose*

O *Docker Compose* [29] é uma ferramenta que permite definir e orquestrar aplicações multi-container (aplicações que necessitam de mais que uma "peça", por exemplo, uma API por norma também necessita de uma BD). Para criar um cenário com o *Kong API Gateway* foi necessário criar um ficheiro *docker-compose* que contêm todas as peças necessárias para o mesmo, bem como algumas peças adicionais para facilitar a configuração do gateway, apresentadas a seguir:

- Networks
  - Kong-net
- Services
  - Kong (API Gateway)
  - Db (Postgres Database)
  - Konga (Kong admin API GUI)

Nesta configuração foi criada uma rede onde todos os serviços vão comunicar entre si. Estes serviços são compostos pelo Kong [20], uma base de dados PostgresSQL [11] para guardar a configuração do Kong, e o Konga [30], um projeto *open-source* que fornece uma GUI para configurar o Kong. No caso do Konga e da BD PostgresSQL foram utilizadas as imagens oficiais para a criação dos serviços, no caso do Kong foi criado um Dockerfile que contém não só a imagem oficial do Kong, mas também tem o processo de instalação de *plugins* necessários.

5.4.2 *Kong API Gateway*

Kong API Gateway é um gateway *open-source* que suporta desenvolvimento *cloud native*. Este age como um *reverse proxy*, o que lhe permite gerir, configurar e encaminhar pedidos para as respetivas API. O Kong contém também vários *plugins* para diversas funcionalidades. Muitos destes *plugins* são oficiais, mas também existem *plugins open-source* desenvolvidos pela comunidade.

No Kong existem duas formas de indicar a configuração de API e serviços. Uma destas formas é através da API de administrador, com acesso a uma BD. Outra alternativa é através de um ficheiro YAML (Yet Another Markup Language) que contém todas as configurações necessárias (sem BD). Inicialmente foi utilizada a abordagem sem BD para configurar todo o processo necessário, mas foi eventualmente discutido que ter uma interface gráfica para criar ou alterar as configurações em *runtime* seria algo importante para a empresa. Sendo assim, foi então decidido utilizar uma abordagem que usa a API de administrador do Kong, em conjunto com um projeto open-source chamado Konga [30], que é uma GUI para a admin API do Kong.

Na configuração do gateway foram criados dois serviços, apresentados na tabela 7.

Serviço	Endpoint no Gateway	Redireciona para endpoint
XtpAuthService	[gateway-ip]:[gateway-port]/xtpauth	[xtpAuth-ip]:[xtpAuth-port]
XtpDriveService	[gateway-ip]:[gateway-port]/xtpdrive	[xtpDrive-ip]:[xtpDrive-port]

Tabela 7: Serviços criados no gateway, com o caminho para o qual o endereço indicado é redirecionado.

O serviço *XtpDriveService* é o serviço que corresponde à *XtpDriveAPI*. Este serviço contém não só a configuração necessária para fazer o redirecionamento para a API, como também um *plugin* que faz o processo de validação da autenticação com JWT [19]. Para a configuração do JWT é necessário indicar a chave pública utilizada pela *XtpAuthAPI*, apresentada na secção 5.1. Depois desta configuração é também possível utilizar o *plugin open-source jwt-auth-rbac* [31] que permite fazer o processo de RBAC, com base no JWT recebido. Neste projeto este *plugin* foi apenas investigado para trabalho futuro. O serviço *XtpAuthService* é o serviço correspondente à *XtpAuthAPI*. Este serviço não contém nenhum *plugin*, sendo assim possível aceder ao mesmo sem qualquer restrição.

Na Figura 29 podem-se ver representados estes serviços, e como estes operam de forma esquemática. A vermelho pode-se o fluxo de um pedido HTTP ao Kong com caminho de */xtpauth* e que este é redirecionado para o endereço *http://xtpauthapi/*, conforme configurado no serviço. A azul pode-se ver o fluxo de um pedido HTTP ao Kong com caminho de */xtpdrive* e que este é redirecionado para o endereço *http://xtpdriveapi/*, conforme configurado no serviço.

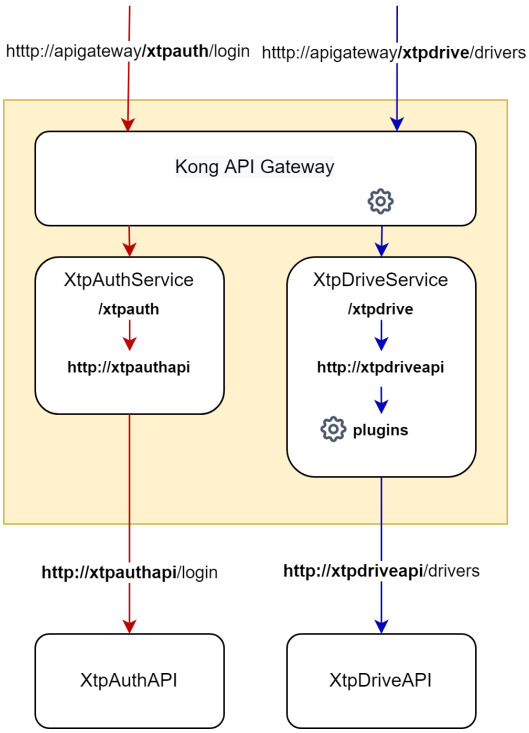


Figura 29: Fluxo de um pedido HTTP ao *Kong API Gateway*

5.5 APLICAÇÃO MÓVEL XTPDRIVEMOBILE

A aplicação móvel desenvolvida no âmbito deste estágio tem como principal objetivo representar informação sobre dados de eficiência energética, orientada ao condutor. Esta informação vem da *XtpDriveAPI* desenvolvida também no contexto deste estágio, apresentada na secção 5.2. A aplicação móvel também interage com a *XtpAuthAPI* para efetuar o processo de autenticação, esta foi apresentada na secção 5.1. Este projeto também utiliza o *XtpDriveDtos*, que contem todos os DTO utilizados pela *XtpDriveAPI*, e que é apresentado na secção 5.3.

Nesta aplicação móvel existem 3 perfis (*roles*) de utilizadores, sendo estes o perfil de condutor, administrador e ambos condutor e administrador em conjunto. Cada um destes perfis terá ligeiras diferenças nas funcionalidades da aplicação.

Esta aplicação foi desenvolvida em Xamarin Forms para Android e iOS. Foram também utilizadas várias bibliotecas para o auxílio ao desenvolvimento deste projeto. A maioria destas bibliotecas contêm implementações de elementos para a UI; no entanto, algumas bibliotecas utilizadas são importantes e devem ser mencionadas.

Neste projeto foi utilizada a biblioteca *FreshMvvm* [35] que facilita bastante o desenvolvimento seguindo o padrão MVVM. Com esta biblioteca, nomes dos ficheiros devem seguir uma *naming convention* em que os nomes dos ficheiros devem acabar em *Page* e *PageModel* em vez de *View* e *ViewModel*. Cada *PageModel* necessita

também de herdar da classe *FreshBasePageModel*, com esta configuração feita, o projeto tem acesso às seguintes funcionalidades:

- Navegação de *PageModel* para *PageModel*;
- *BindingContext* automático de *Page* para *PageModel*;
- Métodos que facilitam o fluxo do código, por exemplo, adição de uma função *Init()* que é a primeira função a correr depois do construtor em cada *PageModel*, ou a adição de uma função *ReverseInit()* que corre sempre que um *PageModel* volta ao topo da *Navigation Stack*;
- IOC (Inversion Of Control);
- Navegação simplificada em TabbedPages.

Outra biblioteca importante neste projeto é o *Xamarin Essentials*. O *Xamarin.Essentials* [14] é um *package* que já vem instalado por omissão, e que fornece funcionalidades específicas para Android e iOS numa API *cross-platform*. É este *package* que permite que seja simples utilizar, por exemplo, o GPS (Global Positioning System) ou a gestão de preferências.

Por fim também é utilizado o *XtpDriveDtos*, utilizado para importar as classes DTO utilizadas nos pedidos e respostas HTTP feitos à *XtpDriveAPI*. Este é o projeto apresentado na secção 5.3

Nos subcapítulos seguintes vão ser apresentadas as principais funcionalidades e mecanismos desenvolvidos nesta aplicação.

5.5.1 *Acesso às API*

Para facilitar acesso às API foi criada uma classe *ApiManager*, que implementa duas interfaces, responsável pela implementação de todos os métodos que sejam pedidos a uma API. Uma destas interfaces é a *IAuthenticationService* que contém um método para cada *endpoint* da *XtpAuthAPI*. A outra interface é a *IEcsmService* que contém um método para cada *endpoint* da XtpDriveAPI.

É de notar que, como já mencionado, todos os pedidos que necessitam de autenticação, têm de receber um JWT no cabeçalho de *Authentication*. Para isto, quando o utilizador faz login na aplicação, o JWT resultante disto é guardado em memória. Posteriormente, sempre que é feito um pedido a uma API que necessite de autenticação, é inicialmente verificado se o JWT que o utilizador tem guardado está expirado. Caso este tenha validade, o JWT é simplesmente utilizado no pedido. Caso este não tenha validade é feito um pedido ao *endpoint /refresh* da *XtpAuthAPI*, para tentar dar *refresh* ao JWT. Caso este *refresh* seja efetuado com sucesso, o

pedido HTTP inicial é executado. Caso o *refresh* não tenha sucesso por algum motivo, o utilizador é redirecionado para o ecrã de login.

5.5.2 *Autenticação (Login, Logout e manter sessão)*

O ecrã de login é onde o utilizador faz o processo de autenticação através de um *username* e password, representado na Figura 30.

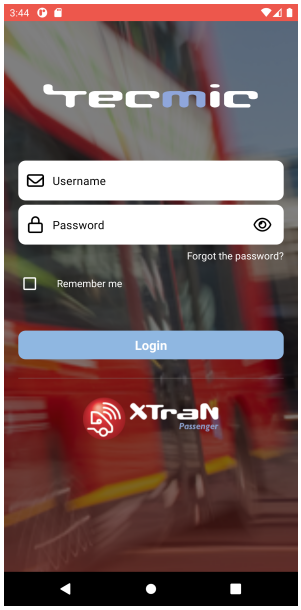


Figura 30: Ecrã de login

Para efetuar o processo de autenticação é utilizado o serviço de autenticação *XtpAuthAPI* ao invocar o *endpoint /signin*. Assim que o utilizador faz login com sucesso é-lhe atribuído um JWT e um *RefreshToken*. Este JWT é guardado nas preferências do utilizador (*SharedPreferences* em Android e *NSUserDefaults* em iOS) para posteriormente ser enviado no cabeçalho *Authorization* dos pedidos HTTP feitos à *XtpDriveAPI*, com o intuito de recolher dados acerca de eficiência energética. Para isto é utilizada a biblioteca *Xamarin Essentials* que abstrai as diferentes implementações de preferências para iOS e Android.

Caso o utilizador selecione a *checkbox Remember me* a sua sessão será mantida, ao reutilizar o JWT guardado nas preferências sempre que este abre a aplicação. Caso o utilizador não selecione a *checkbox*, ao abrir a aplicação, será apresentado o ecrã de login ao utilizador.

5.5.3 *Dashboard*

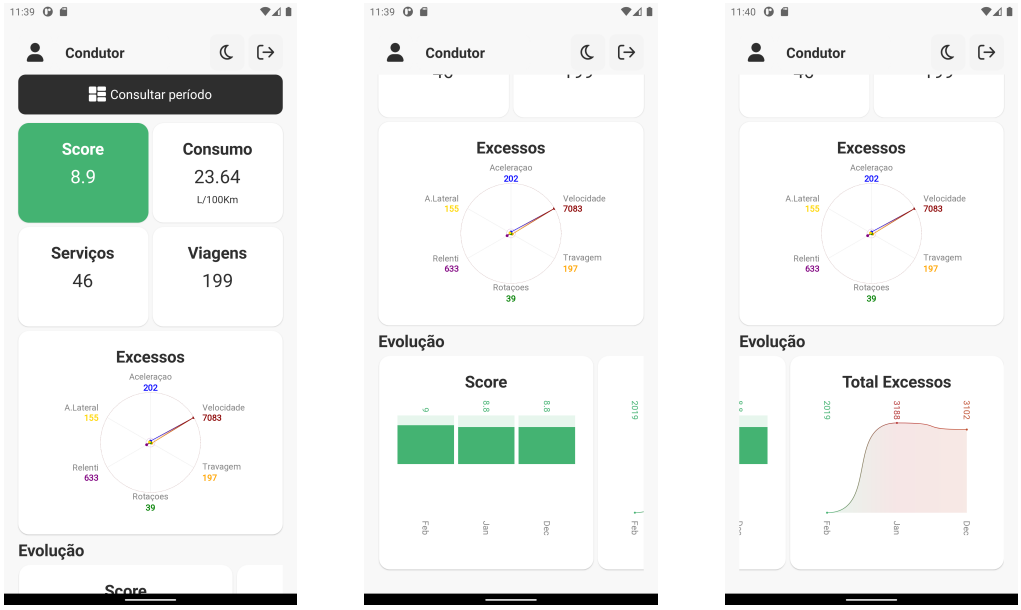
O dashboard é um dos principais ecrãs da aplicação. Este contém estatísticas acerca do condutor nos últimos 6 meses. Estas estatísticas incluem:

- *Score* (valor de 0-10 que representa o desempenho do condutor);
- Consumo médio;
- Número de serviços;
- Número de viagens;
- Um gráfico radar que mostra os diferentes tipos de excessos que o condutor cometeu nos últimos 6 meses.

Este dashboard contém dois gráficos que representam a evolução do condutor nos últimos 6 meses:

- Um gráfico de barras que contém a evolução do *score* nos últimos 6 meses;
- Um gráfico de linhas que contém a evolução do total de excessos nos últimos 6 meses.

Para além disto, o dashboard tem também botões que deixam o utilizador fazer logout e dar *toggle* ao *dark mode*. Existem 3 variantes do dashboard, um para cada tipo de utilizador. Caso o utilizador seja apenas condutor, este apenas pode ver as suas estatísticas, como representado na Figura 31.



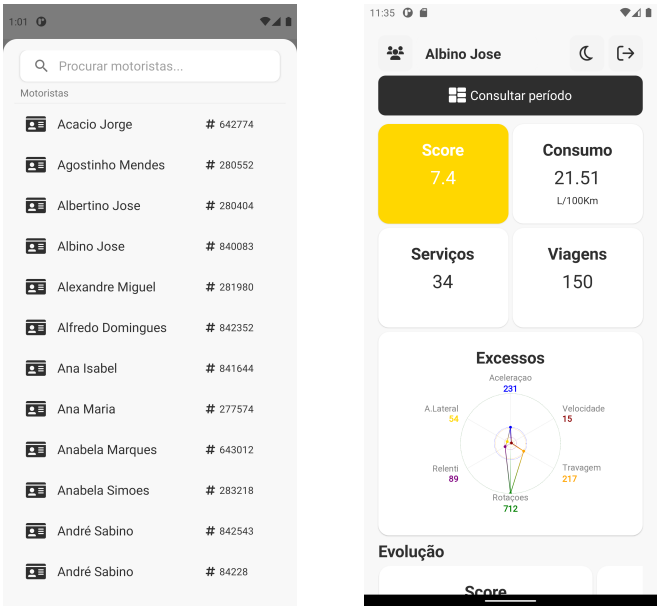
(a) Estatísticas

(b) Gráfico da evolução do score

(c) Gráfico da evolução do total de excessos

Figura 31: Dashboard de um condutor

Caso o utilizador seja apenas administrador, este não tem estatísticas. Assim sendo, este pode apenas selecionar um condutor para o qual planeia consultar estatísticas, sendo inicialmente mostrado o ecrã de seleção de utilizador, em vez do dashboard. Na Figura 32 pode-se ver ambos estes ecrãs.

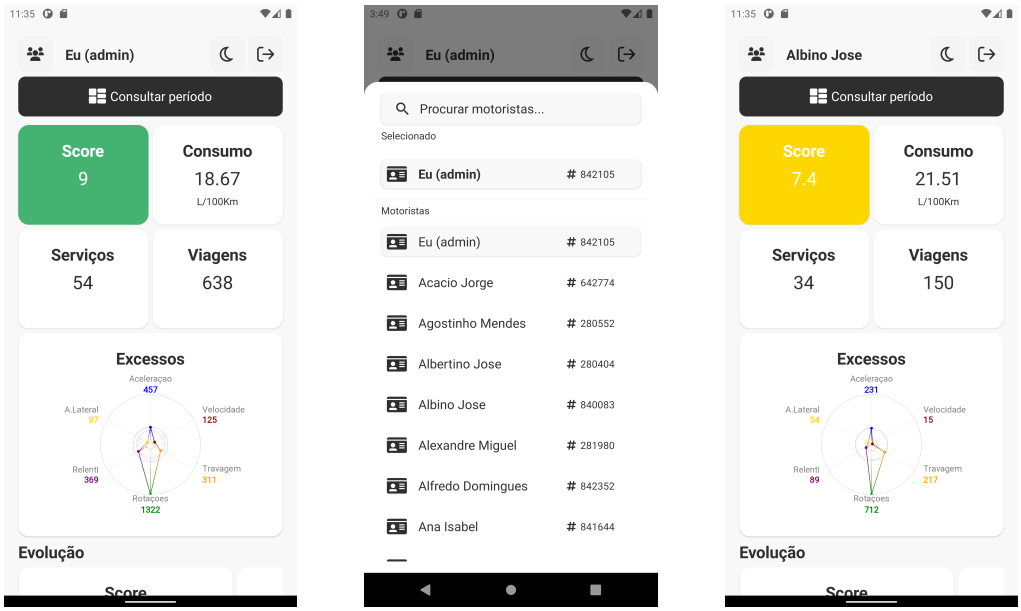


(a) Escolha de condutor

(b) Dashboard de um condutor selecionado

Figura 32: Dashboard de um administrador

Caso o utilizador seja administrador e também condutor, este pode ver tanto as suas estatísticas, como também observar as estatísticas de outros condutores, ao alterar o condutor selecionado, através do botão no canto superior esquerdo com um *icon* de um grupo de pessoas. Na Figura 33 pode-se ver este fluxo do dashboard de um administrador que também é condutor.



(a) Dashboard (b) Escolha de condutor (c) Dashboard de um condutor selecionado

Figura 33: Dashboard de um administrador que também é condutor.

5.5.4 *Filtros/Configuração da consulta*

O ecrã de configuração da consulta pode ser acedido através do dashboard, ao clicar no botão *Consultar Período*. Este ecrã tem diversas opções para o utilizador poder filtrar apenas informação que seja relevante para o mesmo, como representado na Figura 34. É possível filtrar os dados pelos seguintes campos:

- Intervalo de tempo:
  - Última semana;
  - Último mês;
  - Intervalo customizado.
- Selecionar apenas dados em viagem ou não;
- Agrupar por:
  - mês;
  - semana;
  - dia.
- Ordenar por:
  - Data;
  - Distância;

– Consumo;

– Total Excessos.

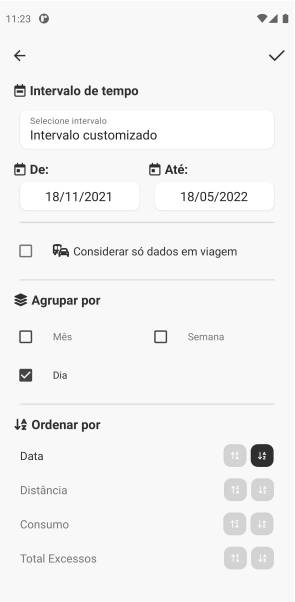


Figura 34: Ecrã de filtros/configuração da consulta

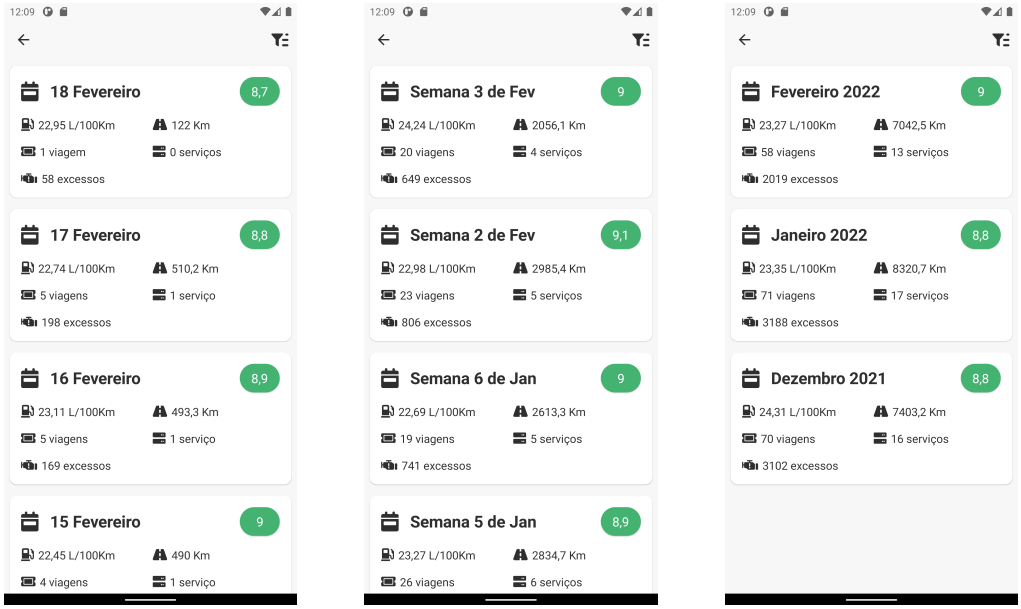
5.5.5 *Lista de períodos de condução*

Este é o ecrã que mostra a lista de períodos de condução com base nas opções selecionadas no ecrã de filtros. Esta lista é composta por uma *CollectionView*, uma vista otimizada para apresentar dados em modo de lista, e que facilita o processo de carregamento de novos registos ao longo do *scroll*. Neste caso, sempre que os últimos 3 registos comecem a aparecer no ecrã do utilizador, são carregados novos dados, com uma nova chamada à *XtpDriveAPI*.

Cada período de condução é representado pelos seguintes campos:

- Data;
- Consumo médio;
- Distância;
- Número de viagens;
- Número de serviços;
- Número total de excessos.

O campo de data varia conforme o filtro selecionado, podendo este ser dia, semana ou mês, conforme representado na Figura 35.



(a) Períodos de condução agrupado por dia

(b) Períodos de condução agrupado por semana

(c) Períodos de condução agrupado por mes

Figura 35: Períodos de condução com diferentes agrupamentos selecionados nos filtros

5.5.6 *Detalhe de período de condução*

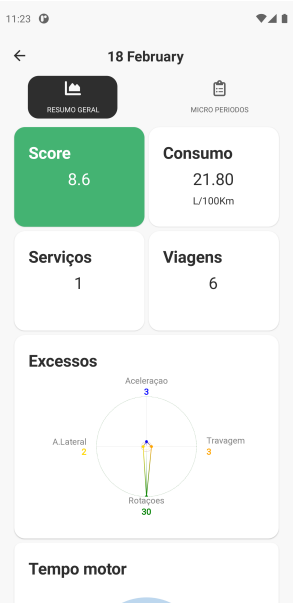
O ecrã de detalhe de período de condução é apresentado assim que o utilizador clica num *item* da lista de períodos de condução. Este ecrã contém 2 *tabs*, que serão descritas de seguida.

5.5.6.1 *Tab 1 - Resumo Geral*

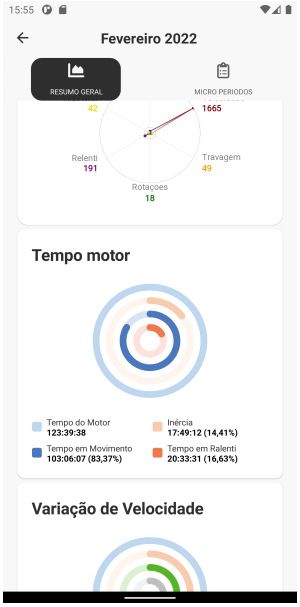
A primeira *tab* contém um resumo geral sobre o período selecionando, com diversas estatísticas, demonstrado na Figura 36. Estas estatísticas têm em conta os seguintes dados:

- Score;
- Consumo médio;
- Número de serviços;
- Número de viagens;
- Um gráfico radar que mostra os diferentes tipos de excessos que o condutor cometeu no período selecionado;
- Um gráfico de *gauge* com dados do tempo do motor. Este gráfico contém:
  - tempo total do motor;
  - tempo em movimento;

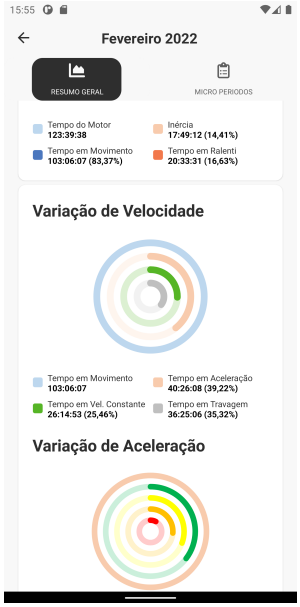
  - tempo em ralenti;
  - tempo em inércia.
- Um gráfico de *gauge* com dados sobre a variação de velocidade. Este gráfico contém:
  - tempo total em movimento;
  - tempo em aceleração;
  - tempo em velocidade constante;
  - tempo em travagem.
- Um gráfico de *gauge* com dados sobre a variação de aceleração. Este gráfico contém:
  - tempo total em aceleração;
  - tempo em aceleração equilibrada;
  - tempo em aceleração moderada;
  - tempo em aceleração indesejável;
  - tempo em aceleração excessiva.
- Um gráfico de *gauge* com dados sobre a variação de travagem. Este gráfico contém:
  - tempo total em travagem;
  - tempo em travagem equilibrada;
  - tempo em travagem moderada;
  - tempo em travagem indesejável;
  - tempo em travagem excessiva.
- Um gráfico de *gauge* com dados sobre a velocidade. Este gráfico contém:
  - tempo total em movimento;
  - tempo em velocidade alta excessiva;
  - tempo em velocidade alta moderada;
  - tempo em velocidade alta indesejável;
  - tempo em velocidade equilibrada;
  - tempo em velocidade baixa indesejável;
  - tempo em velocidade baixa moderada;
  - tempo em velocidade baixa excessiva.



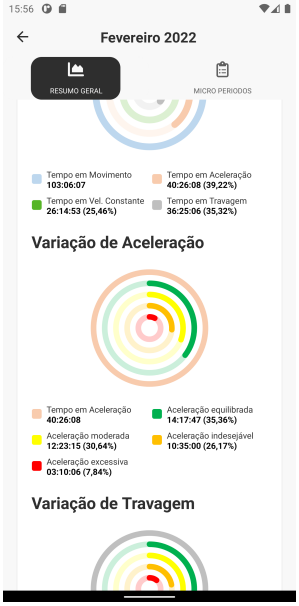
(a) Estatisticas do resumo geral



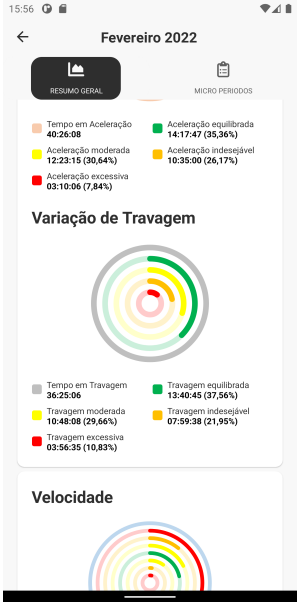
(b) Grafico de gauge com o tempo do motor



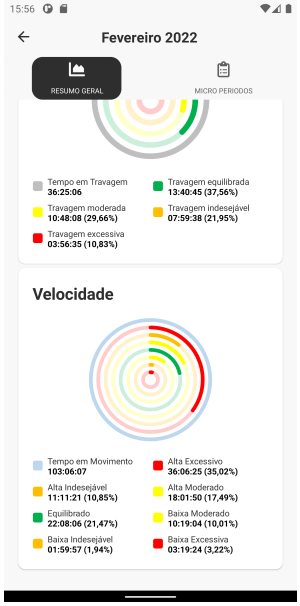
(c) Grafico de gauge com a variação de velocidade



(d) Grafico de gauge com a variação de aceleração



(e) Grafico de gauge com a variação de travagem



(f) Grafico de gauge com a velocidade

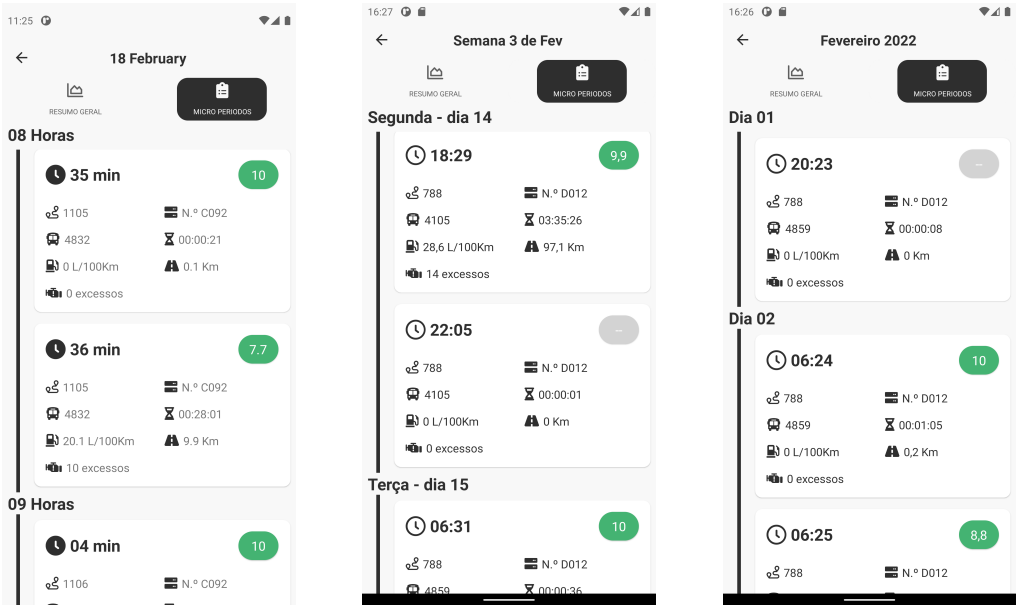
Figura 36: Tab 1 - Resumo geral

5.5.6.2 *Tab 2 - Micro-períodos de condução*

A segunda *tab* contém uma lista de todos os micro-períodos que formam o período selecionado, demonstrado na Figura 38. Cada micro período é representado pelos seguintes campos:

- Número da linha;
- Número do serviço;

- Número de veículo;
- Duração;
- Consumo médio;
- Distância;
- Total de excessos.



(a) Micro-períodos de condução com os períodos de condução agrupados por dia

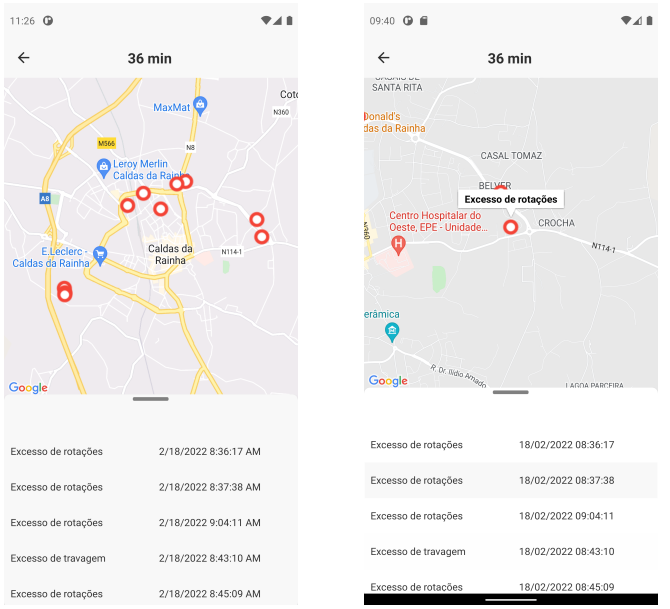
(b) Micro-períodos de condução com os períodos de condução agrupados por semana

(c) Micro-períodos de condução com os períodos de condução agrupados por mes

Figura 37: Tab 2 - Micro-períodos de condução

5.5.7 *Localização dos excessos de um micro-período*

Este ecrã é apresentado assim que o utilizador seleciona um dos micro-períodos da lista. Este ecrã contém um mapa com todos os excessos cometidos, bem como uma lista descritiva dos mesmos. Ao clicar num *item* da lista, o mapa foca no excesso selecionado. Ao clicar num *marker* no mapa, a lista foca no excesso selecionado.



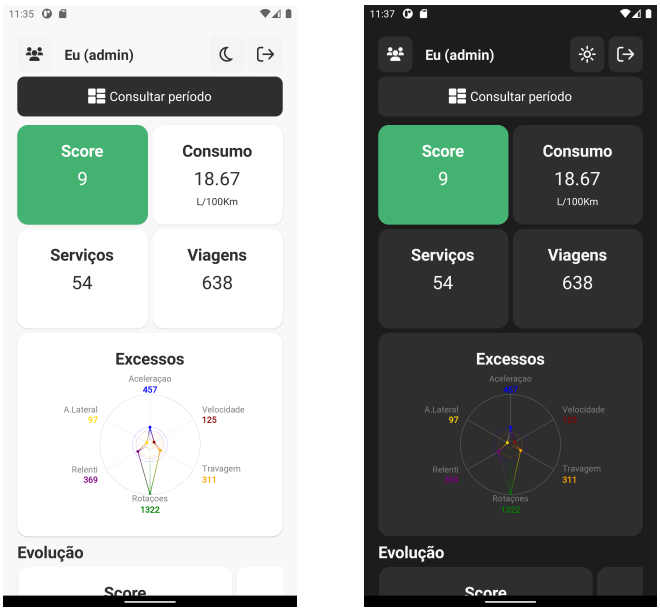
(a) Ecrã com localização dos excessos de um micro-período

(b) Ecrã com localização dos excessos de um micro-período, com um item selecionado

Figura 38: Localização dos excessos de um micro-período

5.5.8 *Temas claro e escuro*

Nesta aplicação foram também implementados 2 temas diferentes, um tema claro e um tema escuro. O principal objetivo desta funcionalidade foi provar o conceito que é possível no futuro ter diferentes temas para diferentes clientes. Na Figura 39 pode se ver uma comparação entre o dashboard nos 2 temas. Este pode ser alterado através do botão no topo do dashboard que contém uma lua/sol conforme o tema selecionado.

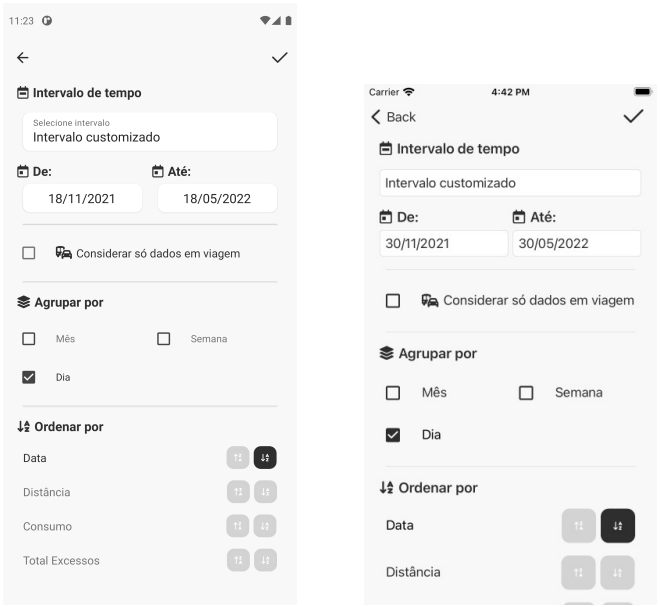


(a) Tema claro (b) Tema escuro

Figura 39: Comparação dos diferentes temas no dashboard

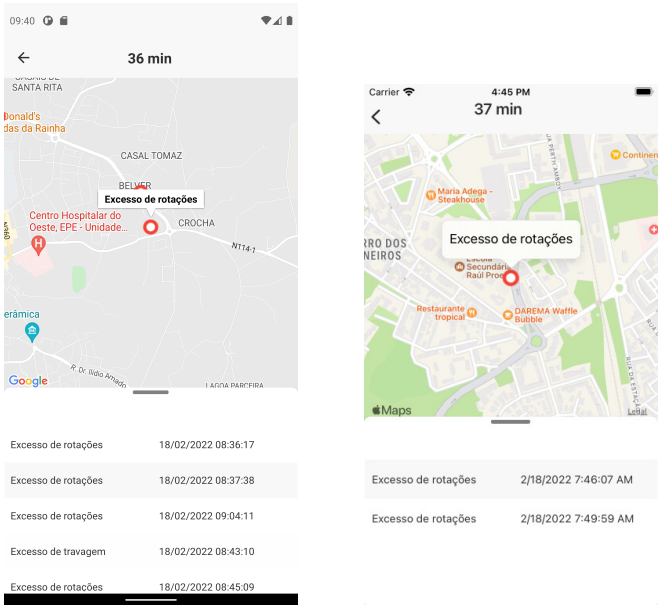
5.5.9 *Comparação Android e iOS*

Como esta aplicação foi desenvolvida em Xamarin Forms, é possível compilar esta para Android e iOS; no entanto, algumas partes da UI são ligeiramente diferentes. Nas figuras 40 e 41 pode-se ver a comparação de alguns ecrãs que são ligeiramente diferentes.



(a) Ecrã de filtros em Android (b) Ecrã de filtros em iOS escuro

Figura 40: Comparação do ecrã de filtros em Android e iOS



(a) Ecrã do mapa de excessos em Android

(b) Ecrã do mapa de excessos em iOS escuro

Figura 41: Comparação do ecrã de localização dos excessos em Android e iOS

5.6 CI/CD

Para todos os projetos apresentados nesta tese foram implementados pipelines no *Azure Pipelines*. Existem dois tipos de pipelines no *Azure Pipelines*: pipelines de *build* e pipelines de *release*. Estes pipelines são compostos por blocos de configurações que existem no *Azure Pipelines*. Estes blocos são ofuscados de como funcionam internamente.

Os pipelines de build têm como principal objetivo certificar-se que, sempre haja uma alteração no código, tudo continua a funcionar, levando assim a que seja mais difícil entrar código não funcional nos respetivos repositórios. Caso sejam encontrados problemas ao correr o pipeline, o código vindo de um *pull request* é recusado automaticamente. Caso não sejam encontrados problemas, este tipo de pipelines por norma cria um artefacto, que é essencialmente o executável de uma aplicação.

Os pipelines de *release* visam automatizar o processo de distribuição das respetivas aplicações, com base nos artefactos gerados.

5.6.1 *XtpAuthAPI e XtpDriveAPI*

Para ambos estes projetos foi utilizado um pipeline de *build* e um pipeline de *release*.

O pipeline de build, foi criado para manter a qualidade do código, ao correr sempre que um *pull request* para o *branch* master fosse feito. É de notar que no caso da XtpDriveAPI são gerados 2 artefactos em caso de sucesso, pois a solução (*dotnet solution*) deste projeto contem ambos o *XtpDriveAPI* e o *XtpDriveDtos*, logo são criados artefactos para cada projeto. Este pipeline, apresentado na Figura 42, é composto pelos seguintes blocos:

- *NuGet restore* — bloco utilizado para importar todos os packages necessários ao projeto;
- *dotnet build* — bloco utilizado para fazer build do projeto;
- *dotnet test* — bloco utilizado para correr todos os testes unitários do projeto;
- *dotnet publish* — bloco utilizado para gerar o artefacto resultante;
- *Publish Artifact* — bloco utilizado para publicar o artefacto no *Azure Artifacts*;

Na Figura 42 pode-se ver a representação deste pipeline no *Azure Pipelines*.

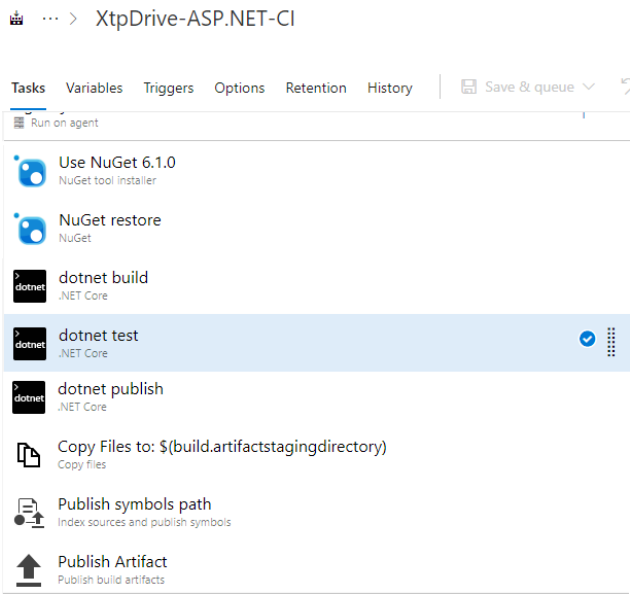


Figura 42: Pipeline de *Build* utilizado nas API — ordem de cima para baixo

O pipeline de *release*, foi criado com o intuito de distribuir as API num servidor de desenvolvimento, composto por um *Windows Server* com IIS (Internet Information Services) instalado. Na configuração do pipeline é indicado de onde vem o artefacto a ser distribuído, neste caso este vem do pipeline de build acima descrito. Este pipeline, apresentado na Figura 43, é composto pelos seguintes blocos:

- IIS Web App Management [37] — bloco utilizado para criar ou atualizar uma *Web App* ou *Application Pool* no IIS;
- IIS Web App Deploy [36] — bloco utilizado para distribuir o artefacto selecionado na máquina alvo, com o *Website Name* indicado;

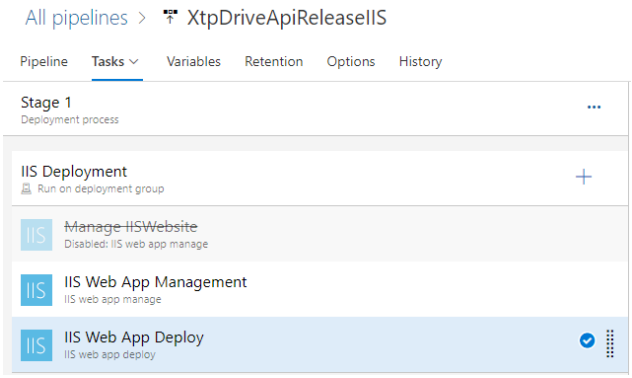


Figura 43: Pipeline de *Release* utilizado nas API — ordem de cima para baixo

5.6.2 *XtpDriveDtos*

O *XtpDriveDtos* e um *NuGet Package*, logo o seu processo de distribuição difere do das API. Para isto foi necessário criar um *Feed* no *Azure Artifacts*, que é essencialmente um repositório onde se podem distribuir NuGet Packages privados. Após a criação e configuração deste *Feed*, foi então criado o pipeline de *release* deste projeto. Este pipeline é constituído por apenas um bloco *NuGet Push*, apresentado na Figura 44. Este bloco publica um artefacto do tipo *NuGet Package* no *Feed* especificado.

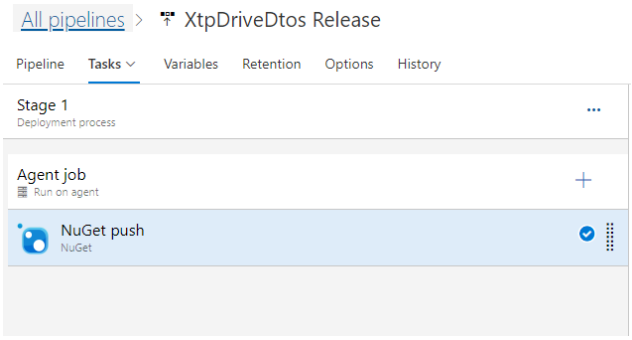


Figura 44: Pipeline de release do XtpDriveDtos

5.6.3 *XtpDriveMobile*

Para o *XtpDriveMobile* não foram implementados pipelines de *release*. No entanto, foi implementado um pipeline de build que é bastante idêntico ao pipeline de build das API. Como o *XtpDriveMobile* é um projeto em Xamarin Forms, este não pode executar o comando *dotnet build*, sendo necessário fazer o build de toda a *solution*, composta pelos projetos específicos para Android e iOS. Como este projeto não terá um pipeline de release, não é necessário criar um artefacto resultante do mesmo. Este pipeline, apresentado na Figura 45, é composto pelos seguintes blocos:

- *NuGet restore* — bloco utilizado para importar todos os packages necessários ao projeto;
- *Build Solution* — bloco utilizado para construir toda a solução, isto inclui os 3 projetos (Android, iOS e Xamarin Forms);
- *dotnet test* — bloco utilizado para correr todos os testes unitários do projeto;

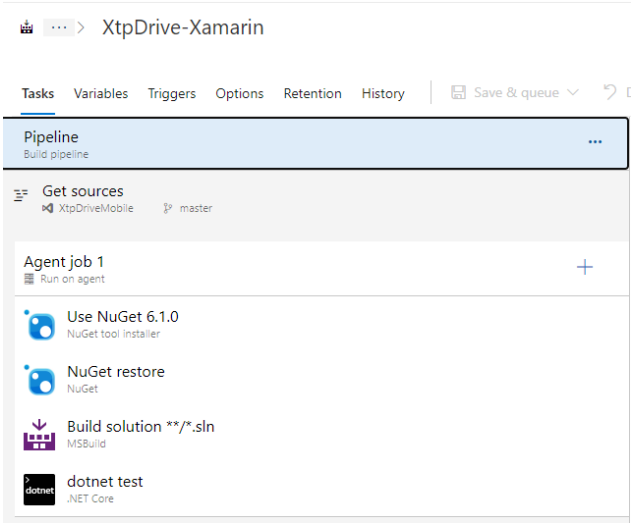


Figura 45: Pipeline de build do XtpDriveMobile

6

TESTES

Testes de software pretendem avaliar e verificar que o software a ser testado funciona como definido, prevenido assim bugs com o desenvolvimento contínuo. Existem diferentes tipos de testes, tais como: testes unitários, testes de integração, testes de aceitação, testes de usabilidade, etc. Neste projeto apenas foram feitos testes unitários e testes de integração, apresentados nas secções seguintes.

6.1 TESTES UNITÁRIOS

Testes unitários são testes que visam testar uma funcionalidade isolada, para verificar que esta funciona como devido. Estes por norma são compostos por múltiplos *test cases* em que são dados vários *inputs*, e os *outputs* que estes devem produzir respetivamente.

Neste projeto foram realizados 67 testes unitários no *XtpDriveMobile*, divididos pelas funcionalidades de cada *ViewModel*. Na *XtpDriveAPI* foram realizados 10 testes unitários, onde foram testadas todas as funções de cada controlador.

Um exemplo de um teste unitário, realizado no XtpDriveMobie, verifica que a função *GetWeekOfMonth* retorna o número da semana correto, dada uma data de *input* e o número da semana esperado, representado na Figura 46.

```
[Theory]
[MemberData(nameof(ExtensionTestsData.GetWeekOfMonthValidParametersTestData), MemberType = typeof(ExtensionTestsData))]
0 references
public void GetWeekOfMonth_ShouldReturnCorrectWeekNumber_WhenGivenADatetime(DateTime inputDate, int expectedWeekNumber)
{
    Assert.Equal(expectedWeekNumber, inputDate.GetWeekOfMonth());
}
```

Figura 46: Exemplo de um teste unitário no *XtpDriveMobile*

6.2 TESTES DE INTEGRAÇÃO

Testes de integração são testes onde diversas componentes e módulos são testados em conjunto. O principal objetivo deste tipo de testes é verificar que uma aplicação funciona em conjunto com todos os seus componentes.

Neste projeto apenas foram feitos testes de integração no XtpDriveAPI, com o intuito de verificar que esta funciona em conjunto com todos os seus componentes.

Um exemplo deste tipo de teste realizado na *XtpDriveAPI*, verifica que o *endpoint /DriverEcsmResults/{mecNumber}* retorna dados válidos. Com este teste verifica-se não só a funcionalidade do controlador, mas também a sua ligação à BD, e ainda que a verificação do JWT passado no cabeçalho Authorization está a funcionar como devido. Este teste está representado na Figura 47.

```
[Theory]
[InlineData("admin", "123456", "275123", HttpStatusCode.OK)]
[InlineData("gestor", "123456", "275123", HttpStatusCode.OK)]
[InlineData("condutor", "123456", "275123", HttpStatusCode.OK)]
0 references
public async Task GetEcsmResults_ShouldReturnData_WhenGivenAValidUser(string username,
    string password, string mecNr, HttpStatusCode expectedStatusCode)
{
    var loginUser = new UserLoginResource
    {
        Username = username,
        Password = password
    };

    var auth = await GetAuthToken(loginUser);

    var client = GetHttpClient();
    client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", auth.AuthorizationToken);

    var result = await client.GetAsync($"DriverEcsmResults/{mecNr}");

    Assert.Equal(expectedStatusCode, result.StatusCode);

    var json = await result.Content.ReadAsStringAsync();
    Assert.NotNull(json);

    var ecsmResults = JsonConvert.DeserializeObject<PaginatedResponse<EcsmResultDto>>(json);
    Assert.True(ecsmResults.Data.Any());
}
```

Figura 47: Exemplo de um teste de integração na XtpDriveAPI

6.3 TESTES NESTE PROJETO

Para a realização dos diferentes testes foi utilizada a biblioteca xUnit [12], que é uma biblioteca *open-source* para testes em .NET e Xamarin Forms.

Foram também utilizadas diversas bibliotecas de *Mock* para simular as dependências das diversas unidades testadas, para que estas sejam testáveis. As bibliotecas utilizadas foram: Moq [25], Moq.Contrib.HttpClient [23], e Xamarin.Forms.Mocks [15].

Na Figura 48 pode-se ver um exemplo do *Mock* da dependência *IConfiguration*. Neste exemplo é definido que quando a chave *[“XtpDriveApiUrl“]* for invocada, irá ser retornado o valor da *string “xtpdriveUrl“*, e quando a chave *[“XtpAuthApiUrl“]* for invocada, irá ser retornado o valor da *string “xtpauthUrl“*.

```
private readonly Mock<IConfiguration> configMock = new Mock<IConfiguration>();

private string xtpdriveUrl = "http://192.168.2.21:8000/xtpdrive";
private string xtpauthUrl = "http://192.168.2.21:8000/xtpauth";

0 references
public ApiDriveManagerTests()
{
    configMock.Setup(x => x["XtpDriveApiUrl"]).Returns(xtpdriveUrl);
    configMock.Setup(x => x["XtpAuthApiUrl"]).Returns(xtpauthUrl);

}
```

Figura 48: Exemplo da simulação de uma dependência do tipo IConfiguration

Desta forma, as classes que dependem de uma *IConfiguration* passam a ser testáveis, pois os métodos necessários são simulados com o que estes retornam.

Em xUnit existem dois tipos de teste: *Fact* e *Theory*. A principal diferença entre os dois é que os testes *Theory* têm a opção de serem parametrizados, enquanto os *Facts* não.

Na Figura 49 pode-se ver um exemplo de um teste do tipo *Fact* em xUnit que verifica que quando se executa o *ICommand OnClick*, a função *CoreMethods.PushPageModel* é chamada, verificando assim a navegação correta da aplicação. Ainda neste exemplo podemos verificar que é simulada da dependência *CoreMethods* do *ViewModel* testado. Nesta simulação apenas verificamos que o método *PushPageModel* é chamado uma vez.

```
[Fact]
0 references
public void OnClick_ShouldPushNavigationProperly_WhenCalled()
{
    var page = new MainPageModel(authServiceMock.Object, ecsmServiceMock.Object, preferencesMock.Object);
    page.CoreMethods = mockCoreMethods.Object;

    page.OnClick.Execute(null);

    mockCoreMethods.Verify(c => c.PushPageModel<FilterPageModel>(It.IsAny<Action<FilterPageModel>>(),
        false, true), Times.Once);
}
```

Figura 49: Exemplo de um teste do tipo *Fact* usado no *XtpDriveMobile*.

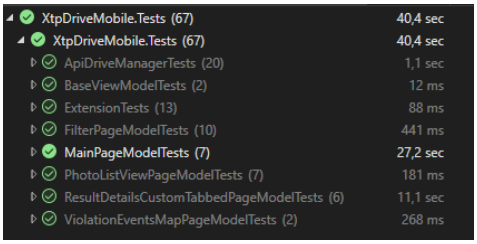
Na Figura 50 pode-se ver um exemplo de um teste do tipo *Theory* em xUnit que tem como entrada duas *strings*, sendo a primeiro o input a testar, e a segunda o resultado esperado.

```
[Theory]
[InlineData("teste", "Teste")]
[InlineData("TESTE", "Teste")]
[InlineData("tESTE", "Teste")]
[InlineData("", "")]
[InlineData(null, "")]
[InlineData("t", "T")]
0 references
public void FirstCharToUpper_ShouldReturnStringWithFirstCharacterUpper_WhenGivenAnyString(string input, string output)
{
    Assert.Equal(output, input.FirstCharToUpper());
}
```

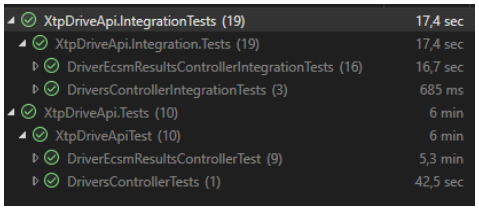
Figura 50: Exemplo de um teste do tipo *Theory* usado no *XtpDriveMobile*.

No total foram feitos 67 testes unitários no *XtpDriveMobile* divididos pelos diferentes *ViewModels*, representados na Figura 51. Foram também feitos 10 testes unitários na *XtpDriveAPI* divididos pelos diferentes controladores, e 19 testes de integração representados na Figura 51.

Estes testes foram feitos incrementalmente, ao longo do processo de desenvolvimento, com o intuito de manter sempre a qualidade do código desenvolvido. Sempre que um teste não passou, o código da funcionalidade a ser testada foi alterado para acomodar o teste, e sempre que uma funcionalidade foi alterada, os testes associados a essa funcionalidade foram também alterados para acomodar a nova funcionalidade.



(a) Testes feitos no XtpDriveMobile



(b) Testes feitos no XtpDriveApi

Figura 51: Testes feitos neste projeto.

6.4 DISCUSSÃO DE RESULTADOS

Os testes realizados neste projeto conseguiram assegurar uma maior qualidade de código. Com os testes unitários conseguiu-se validar todas as principais funções dos projetos, onde foram encontradas diversas falhas que foram corrigidas logo de seguida. Com os testes de integração conseguiu-se provar que todas as componentes da *XtpDriveAPI* estão funcionais.

Apesar disto, não foram realizados testes de usabilidade por uma questão de tempo e disponibilidade do cliente. Estes testes são algo importante para verificar que os protótipos desenvolvidos estão de acordo com o esperado.

7

CONCLUSÕES E TRABALHO FUTURO

O projeto desenvolvido no âmbito do estágio realizado na Tecmic conseguiu alcançar os objetivos propostos. Este projeto fornece assim à Tecmic um protótipo que pode ser apresentado aos seus clientes, de transportes públicos de passageiros, que visam agilizar e facilitar a consulta de dados de eficiência energética, através de uma aplicação móvel. Esta consulta pode incentivar os motoristas da empresa a terem uma condução mais eficiente, ao demonstrar onde estes cometem mais irregularidades e excessos, e ainda o tipo de excessos que estes fazem. Com isto os gestores de frotas também podem observar o desempenho dos seus motoristas, bem como incentivá-los a melhorar a sua condução. Isto leva à redução dos custos da manutenção de veículos, e do consumo dos mesmos.

Tal como apresentado ao longo deste documento, para o desenvolvimento deste projeto foi feita uma análise da ferramenta *XTraN Passenger Web*, onde foram identificadas as principais funcionalidades a extrair para a aplicação móvel *XtpDriveMobile*. Foi também apresentada uma comparação entre as diferentes abordagens de tecnologias de desenvolvimento multi-plataforma e uma visão geral do Xamarin Forms, a tecnologia utilizada neste projeto para o desenvolvimento da aplicação *XtpDriveMobile* para Android e iOS. De seguida foi dada uma perspetiva das várias abordagens na arquitetura de um sistema de micro-serviços, focando no tópico de autenticação destes sistemas. Foi também criada uma arquitetura que fosse escalável ao nível de micro-serviços, mas que também tivesse em conta os sistemas atuais do XTraN Passenger. No trabalho desenvolvido neste projeto, foi criada a aplicação móvel *XtpDriveMobile*, bem como uma infraestrutura que a suportasse. Esta infraestrutura é constituída pela API *XtpAuthAPI*, a API *XtpDriveAPI*, o NuGet Package *XtpDriveDtos*, e uma API Gateway. Esta infraestrutura foi implementada para ser escalável tendo em conta as componentes desenvolvidas e os projetos já existentes na empresa. Apesar de tudo isto, ainda existem funcionalidades a implementar no futuro, que complementem este módulo desenvolvido.

7.1 TRABALHO FUTURO

Como trabalho futuro, existem algumas funcionalidades a implementar no sistema *XTraN Passenger* para acomodar este módulo implementado ao longo do estágio, apresentadas de seguida:

- Criar um módulo de processamento de mensagens, que processa dados vindos do XtpServer e com o auxílio da XtpDriveAPI, através de pedidos HTTP POST, persiste estes dados na BD.
- Adicionar a possibilidade de filtrar dados por linha e veículo (XtpDriveAPI e XtpDriveMobile).
- Criação de testes de usabilidade com os clientes finais, bem como a análise dos resultados obtidos.

BIBLIOGRAFIA

[1] URL: https://cordova.apache.org/.

[2] URL: https://flutter.dev/.

[3] *Agile Manifesto*. URL: https://agilemanifesto.org/iso/ptpt/manifesto.html.

[4] *Azure Devops Services: Microsoft Azure*. URL: https://azure.microsoft.com/en-us/services/devops.

[5] Andreas Biørn-Hansen, Tor-Morten Grønli e Gheorghita Ghinea. «A survey and taxonomy of core concepts and research challenges in cross-platform mobile development». Em: *ACM Computing Surveys* 51.5 (2019), pp. 1–34. DOI: 10.1145/3241739.

[6] Ramaswamy Chandramouli. *Security strategies for microservices-based Application Systems*. Ago. de 2019. URL: https://csrc.nist.gov/publications/detail/sp/800-204/final.

[7] Lisandro Delia et al. «Multi-platform Mobile Application Development Analysis». Em: *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)* (2015). DOI: 10.1109/rcis.2015.7128878.

[8] Muhammad Shoaib Farooq et al. «Cross-Platform Mobile Development Approaches and Frameworks». Em: (2022).

[9] Konrad Gos e Wojciech Zabierowski. «The comparison of Microservice and monolithic architecture». Em: *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)* (2020). DOI: 10.1109/memstech49584.2020.9109514.

[10] *Gravitee.io*. URL: https://www.gravitee.io/.

[11] PostgreSQL Global Development Group. Ago. de 2022. URL: https://www.postgresql.org/.

[12] *Home*. URL: https://xunit.net/.

[13] Ionicframework. *Cross-platform mobile app development*. URL: https://ionicframework.com/.

[14] Jamesmontemagno. *Xamarin Essentials*. URL: https://docs.microsoft.com/en-us/xamarin/essentials.

[15] Jonathanpeppers. *Xamarin.forms.mocks: Library for running Xamarin.forms inside of unit tests.* URL: https://github.com/jonathanpeppers/Xamarin.Forms.Mocks.

[16] M. Jones, J. Bradley e N. Sakimura. *JSON web token (JWT).* Mai. de 1970. URL: https://www.rfc-editor.org/rfc/rfc7519.

[17] JS Foundation - js.foundation. *A Touch-Optimized Web Framework.* URL: https://jquerymobile.com/.

[18] Graham Kaemmer. *Best practices for authorization in microservices.* Dez. de 2021. URL: https://www.osohq.com/post/microservices-authorization-patterns.

[19] KongHQ. *JWT plugin.* URL: https://docs.konghq.com/hub/kong-inc/jwt/.

[20] KongHQ. *Kong gateway - v2.8.x.* URL: https://docs.konghq.com/gateway/latest/.

[21] Mounaim Latif et al. «Cross Platform Approach for Mobile Application Development: A Survey». Em: *2016 International Conference on Information Technology for Organizations Development (IT4OD)* (2016). DOI: 10.1109/it4od.2016.7479278.

[22] Jeffrey A. Livermore. «Factors that significantly impact the implementation of an agile software development methodology». Em: *Journal of Software* 3.4 (2008). DOI: 10.4304/jsw.3.4.31-36.

[23] Maxkagamine. *moq.contrib.httpclient: A set of extension methods for mocking httpclient and ihttpclientfactory with moq.* URL: https://github.com/maxkagamine/Moq.Contrib.HttpClient.

[24] Mjrousos. *Overview of ASP.NET core authentication.* URL: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-6.0.

[25] Moq. *Moq/Moq4: Repo for managing moq 4.X.* URL: https://github.com/moq/moq4.

[26] Sam Newman. *Monolith to microservices: Evolutionary patterns to transform your monolith.* O'Reilly Media, Inc, 2020.

[27] *Ocelot .NET core API Gateway.* URL: https://github.com/ThreeMammals/Ocelot.

[28] Inc. OpenSSL Foundation. *OpenSSL.* URL: https://www.openssl.org/.

[29] *Overview of docker compose.* Ago. de 2022. URL: https://docs.docker.com/compose/.

[30] Pantsel. *Pantsel/Konga: More than just another GUI to kong admin api.* URL: https://github.com/pantsel/konga.

[31] Prabago. *Prabago/kong-plugin-JWT-auth-RBAC: Kong plugin that performs authorization based on custom role claim in JWT.* URL: https://github.com/prabago/kong-plugin-jwt-auth-rbac.

[32] *React native.* URL: https://reactnative.dev/.

[33] Rick-Anderson. *Introduction to identity on asp.net core.* URL: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity.

[34] Rick-Anderson. *Minimal apis overview.* URL: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0.

[35] rid00z. *FreshMvvm.* URL: https://github.com/rid00z/FreshMvvm.

[36] RoopeshNair. *IIS web app deploy task - Azure pipelines.* URL: https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/iis-web-app-deployment-on-machine-group?view=azure-devops.

[37] RoopeshNair. *IIS Web app manage task - Azure pipelines.* URL: https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/iis-web-app-management-on-machine-group?view=azure-devops.

[38] *Scrum.* URL: https://www.scrum.org/.

[39] Sarang Shaikh e Sindhu Abro. «Comparison of traditional and Agile Software Development methodology: A short survey». Em: *International Journal of Software Engineering and Computer Systems* 5.2 (2019), pp. 1–14. DOI: 10.15282/ijsecs.5.2.2019.1.0057.

[40] Tzachi Strugo. *Authentication & Authorization in Microservices Architecture.* Fev. de 2021. URL: https://dev.to/behalf/authentication-authorization-in-microservices-architecture-part-i-2cn0#:~:text=Authentication%5C%20%5C%26%5C%20Authorization%5C%20on%5C%20each%5C%20service,to%5C%20implement%5C%20their%5C%20security%5C%20solution.

[41] *Titanium Mobile.* URL: https://titaniumsdk.com/.

[42] Don Wells. *Extreme programming: A gentle introduction.* URL: http://www.extremeprogramming.org/.

[43] *Xamarin: Open-source mobile app platform for .NET.* URL: https://dotnet.microsoft.com/en-us/apps/xamarin.

[44] Dongjin Yu et al. «A survey on security issues in Services Communication of microservices-enabled Fog Applications». Em: *Concurrency and Computation: Practice and Experience* 31.22 (2018). DOI: 10.1002/cpe.4436.