Mestrado em Engenharia Informática

Estágio

Relatório Final

Multiplatform Apps - sistema de distribuição e visualização de conteúdos multiplataforma

Diogo Reis Falcão

falcao@student.dei.uc.pt

Orientador do DEI-FCTUC:

Professor Doutor Fernando Barros

Orientador da *Ubiwhere*:

Bruno Silva

Coimbra, 1 de Setembro de 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Mestrado em Engenharia Informática

Estágio

Relatório Final

Multiplatform Apps - sistema de distribuição e visualização de conteúdos multiplataforma

Diogo Reis Falcão

falcao@student.dei.uc.pt

Júri:

Professor Doutor António Mendes

Professor Doutor Pedro Furtado

Coimbra, 1 de Setembro de 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

Atualmente, a comunicação empresa-cliente é feita em grande escala através da utilização de dispositivos modernos com ligação à rede, que por sua vez servem para apresentar mensagens digitais. Estas mensagens têm associadas, por norma, contéudos multimédia para promover a sua dinamicidade e, por consequente, aumentar a qualidade de comunicação. Por isso, são ideais para apresentação em lojas de retalho. No caso concreto das lojas de dispositivos, existe uma grande oportunidade para mostrar mensagens, que é os seus próprios ecrãs. Contudo, para fazer proveito destes, é necessário ter em conta o sistema operativo do dispositivo e ainda o método de distribuição do conteúdo multimédia.

Posto isto, este estágio vai ter como objetivo o desenvolvimento de um sistema que permita lidar com a exibição de conteúdos em diferentes sistemas operativos e que possibilite a distribuição destes a partir de um servidor central. Deste modo, vão ser implementadas duas aplicações multiplataforma com recurso a *frameworks* específicas. Estas aplicações destinam-se à visualização de conteúdos e vão ser posteriormente integradas com um servidor de distribuição de conteúdos. O sistema vai permitir que um administrador de loja controle o que está a ser visualizado em qualquer dispositivo exposto, visando também facilitar, automatizar e melhorar o processo de distribuição/exibição de conteúdos.

Palavras-chave: *Framework*, Aplicação multiplataforma , Dispositivos modernos, Conteúdo multimédia, Comunicação

Agradecimentos

A toda a minha família, um muito obrigado por toda a motivação e incentivo, em especial à minha Mãe e ao meu Pai, por me proporcionarem esta oportunidade. À minha namorada, Cláudia Silva, agradeço toda a paciência, dedicação e ajuda prestada para me ajudar a ultrapassar esta etapa.

Aos meus colegas de curso e amigos por toda a disponibilidade e ajuda prestada. Destaco o Francisco Cardoso por todo o companheirismo e ajuda durante todo o nosso estágio.

À *Ubiwhere*, pela oportunidade, disponibilidade, acolhimento e bons momentos partilhados durante este projeto. Em particular, à equipa de Coimbra: André Duarte, Francisco Monsanto, Nuno Khan e Ricardo Vitorino.

Ao Bruno Silva e Professor Doutor Fernando Barros, por toda a orientação, ajuda e disponibilidade que tiveram para comigo.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

AOT	Ahead Of Time
API	Application Programming Interface
CMMI	Capability Maturity Model - Integration
CSS	Cascade Style Sheet
DOM	Document Object Model
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Enviroment
JS	Javascript
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MTV	Model View Template
MVC	Model View Controller
MVVM	Model View ViewModel
NPM	Node Package Manager
ORM	Object-Relational Mapping
PHP	PHP: Hypertext Preprocessor
REST	Representational State Transfer
ROI	Return of Investment
SAAS	Software As A Service
SDK	Software Development Kit
SEI	Software Engineering Institute
TTR	Time To Refresh
UI	User Interface
WAMP	Web Application Messaging Protocol
XML	eXtensible Markup Language

Capítulo 1

Introdução

A comunicação com clientes é frequentemente realizada através da exibição de mensagens em ecrãs digitais. Esta forma de comunicação é conhecida por *Digital Signage* e pode ser encontrada em diversos locais como: transportes, museus, estádios, lojas de retalho, hotéis ou restaurantes. A *Digital Signage* pode ser utilizada para, por exemplo: informar os utilizadores de um aeroporto dos horários de um voo (informativa), fazer publicidade a um produto numa loja de retalho (publicitária) ou entreter um cliente que se encontre à espera de ser atendido (publicitária/informativa).

No caso concreto da comercialização de dispositivos digitais em lojas de retalho, é apresentada uma grande oportunidade para a utilização de *Digital Signage*. Esta oportunidade surge da disponibilidade de uma grande quantidade de ecrãs gratuitos para exibição conteúdos multimédia (mensagens). Assim, tendo em conta a possibilidade de utilização de diferentes dispositivos/ecrãs para exibição de conteúdos, os objetivos deste estágio são:

- Construir uma ou mais aplicações, utilizando *frameworks* de desenvolvimento multiplataforma, cujo principal objetivo é apresentação de conteúdos multimédia nos sistemas operativos *Windows, Mac OS, Linux, Android* e *iOS*;
- Integrar e modificar uma plataforma já existente, utilizada para fazer a gestão e distribuição de conteúdos para dispositivos remotos. Pretende-se que as aplicações multiplataforma obtenham automaticamente desta plataforma um conjunto de ficheiros a exibir e o período de tempo em que estes podem ser visualizados. Assim como, eventuais alterações efetuadas ao conjunto de ficheiros/agendamento já atribuídos em *real-time*.

Estes objetivos têm o intuito de criar um sistema automático de *Digital Signage*, que permita distribuir e controlar a exibição de conteúdos remotos, em diferentes dispositivos.

1.1 Motivação

A exibição de conteúdos multimédia possibilita melhorar a comunicação entre empresa-cliente e representa uma forma dinâmica de transmissão de mensagens[114]. Estas mensagens podem trazer valor acrescido para a empresa ao permitirem: ampliar o grau de satisfação do cliente, divulgar marcas/serviços ou aumentar vendas. Assim, o número de dispositivos em que pode ser visualizado determinado conteúdo torna-se um fator importante para qualquer empresa. Contudo, para exibir conteúdos em múltiplos dispositivos é necessário ter em conta fatores como:

- **Sistema operativo suportado**: dispositivos como *smarthphones* ou *tablet's* utilizam sistemas operativos *mobile* (*iOS, Android*), ao contrário dos portáteis ou computadores pessoais que correm sistemas operativos *desktop* (*Windows, Linux, Mac OS*);
- **Resolução/tamanho de ecrã**: para que a mensagem apresentada tenha a melhor qualidade possível, deve ser adaptada às capacidades de visualização oferecidas pelo dispositivo;
- **Método de distribuição utilizado**: normalmente, os conteúdos são disponibilizados utilizando dispositivos físicos de armazenamento, implicando deslocações e monitorização pessoal do processo.

Estes fatores levam a que a exibição de conteúdos em múltiplos dispositivos se torne num processo complexo em que incorrem custos acrescidos ao nível de recursos como: tempo, dinheiro ou pessoal. Surgindo, assim, a necessidade de criação de um sistema de gestão/exibição de conteúdos, que tenha em conta as diferenças entre dispositivos e que promova a distribuição automática de conteúdos de forma a rentabilizar a utilização de ecrãs digitais como forma de comunicação.

Pretende-se que o sistema melhore e facilite as tarefas de um administrador de loja(s), que deseje utilizar dispositivos expostos para apresentação de multimédia aos seus clientes. Para isto, o sistema a desenvolver, vai permitir:

- A reprodução automática de vídeos nos formatos *MP4, FLV* ou *3GP* e imagens nos formatos *PNG*, *JPG*, *SFG* ou *GIF*, em diferentes sistemas operativos;
- A gestão de conjuntos de ficheiros (*playlists*) e agendamentos que podem ser atribuídos a um dispositivo;
- A gestão dos dispositivos das várias lojas, onde podem ser apresentados conteúdos multimédia;
- A propagação/reação automática a alterações de agendamento, ou *playlists*, para os dispositivos afetados.

Em suma, o objetivo é criar um ecossistema específico para os administradores de lojas de retalho de dispositivos, que permita aproveitar mais eficazmente a quantidade de ecrãs gratuitos disponível; promovendo um desenvolvimento mais vantajoso da comunicação empresa-cliente.

1.2 Contexto de utilização de *frameworks* para desenvolvimento multipaltaforma

Como já referido, vão ser construidas uma ou mais aplicações para as plataformas *Windows, Mac OS, Linux, Android* e *iOS*, em que o seu principal objetivo é a exibição de conteúdos multimédia obtidos remotamente.

A um *software* que existe em mais que um sistema operativo pode ser dado o nome de aplicação multiplataforma. No entanto, para suportar vários sistemas, é por norma, necessário fazer a mesma implementação em linguagens de programação diferentes. No caso das plataformas em que o sistema a desenvolver deve estar presente, teriam de ser utilizadas as linguagens apresentadas de seguida.

Plataforma	**Linguagem**
Windows	C++,C#
Linux	C,C++
Mac OS	Objective-C, Swift
Android	Java
iOS	Objective-C, Swift

Tabela 1.1: Linguagens utilizadas para o desenvolvimento nativo.

Na Tabela 1.1 é possível observar seis linguagens diferentes que podem ser empregadas para o desenvolvimento das aplicações de exibição de conteúdos multimédia, para os sistemas operativos requeridos. Contudo, a utilização destas linguagens, implicaria um processo de aprendizagem e adaptação muito grande e levaria a esforços duplicados no que toca a implementação da mesma funcionalidade para diferentes sistemas operativos. Refletindo-se no aumento da complexidade e tempo de desenvolvimento deste projeto.

Posto isto, as *frameworks* para desenvolvimento multiplataforma vão ser utilizadas de forma a facilitar o processo de implementação da aplicação de exibição de conteúdos, para múltiplos sistemas operativos. Estas *frameworks*, tentam abstrair o ambiente final de execução de um programa, ao possibilitar o uso de uma linguagem de programação única que permite o desenvolvimento de aplicações multiplataforma.

1.3 Possível caso de uso do sistema

Um possível caso de uso do sistema é quando um administrador de uma cadeia de lojas pretende reproduzir a mesma publicidade em vários dispositivos, localizados em lojas diferentes. Sem uma maneira automática de o fazer, este teria de transmitir a cada responsável de loja todas as informações necessárias, mais concretamente, o conteúdo multimédia a exibir, os dispositivos onde deve ser reproduzido e o agendamento da sua exibição. Após a receção destas informações, o responsável de loja precisaria de configurar os dispositivos necessários com o conteúdo multimédia recebido e começar a exibição deste, tendo em conta o agendamento também recebido. Com a inexistência de um sistema automático de distribuição/exibição de conteúdos, o

responsável de loja seria obrigado a ligar uma unidade de armazenamento móvel [1] em cada dispositivo, para fazer a disseminação do conteúdo multimédia. E, após isto, começar a exibição do conteúdo nos agendamentos impostos para cada dispositivo, tornando o processo de exibição da mesma publicidade, em vários dispositivos, demorado e com a necessidade de monitorização física.

Ao utilizar o sistema a desenvolver, o administrador de uma cadeia de lojas pode aceder a uma aplicação web onde são mostrados todos os dispositivos de todas as suas lojas. Após isto, só necessita de criar uma *playlist* com o conteúdo multimédia a exibi, o seu agendamento, e atribuí-la aos dispositivos onde pretende mostrar a publicidade. Na conclusão desta atribuição, os dispositivos vão automaticamente exibir o conteúdo multimédia, tornando, assim, possível a gestão/distribuição de conteúdos remotamente.

1.4 Principais desafios

Os principais desafios deste projeto passam pelo facto de ser necessário construir aplicações, utilizando *frameworks* de desenvolvimento multiplataforma, as quais são tecnologia recente e totalmente desconhecida pelo estagiário. Além disto, é ainda necessária a modificação/integração de um servidor de gestão/distribuição de conteúdos, previamente desenvolvido, com as aplicações responsáveis pela exibição de conteúdos em diferentes sistemas operativos.

Na integração/modificação, os principais desafios serão a necessidade de aprendizagem e configuração do funcionamento do servidor, por forma a possibilitar a comunicação servidor-aplicação multiplataforma.

1.5 A empresa

O estagiário foi inserido na empresa *Ubiwhere* que está sediada em Aveiro e conta com escritórios no IPN em Coimbra e em São João da Madeira.Trata-se de uma empresa de *software* fundada em 2007 que contém uma presença mais forte na área do turismo, telecomunicações, transportes e *smart cities*. A *Ubiwhere* conta actualmente com vários produtos no mercado como o *bikeeemotion* [1], *uParking* [73], *uMeter DTT* [20] e o *TourEmotion* [38].

[1] *USB sticks*, *SD cards* ou *external hard drives*

Capítulo 2

Estado da Arte

Este capítulo tem como objetivo aprofundar alguns conceitos fulcrais já introduzidos, de forma a contribuir para uma melhor compreensão do projeto a desenvolver. Esses conceitos são: *Digital Signage*, aplicação ou *framework* multiplataforma. Além disso, são apresentadas as decisões mais relevantes tomadas em relação às etapas de planeamento e construção do sistema automático de distribuição/exibição de conteúdos, detalhando sistemas semelhantes e *frameworks* consideradas para a implementação.

2.1 *Digital Signage*

Este tipo de informação digital utiliza ecrãs de vários tipos (*hardware*) para apresentar conteúdos multimédia através de *software* específico. Normalmente, a informação deste género é colocada em sítios públicos e é personalizada de forma a mostrar conteúdo apropriado ao local. Os conteúdos apresentados pelos ecrãs são atualizados com elevada frequência e são obtidos da *internet* ou de uma rede local. Ao contrário da publicidade clássica, onde o conteúdo apresentado é estático, este tipo de publicidade permite a apresentação de conteúdos dinâmicos que podem ser alterados em qualquer instante. Isto possibilita chamar a atenção da audiência de uma melhor forma.[109]

2.1.1 Propósitos dos conteúdos de *Digital Signage*

A informação apresentada em qualquer ecrã utilizado pela *Digital Signage* varia consoante a sua localização, a audiência alvo e o tipo de ambiente em que este está inserido. Posto isto, o conteúdo apresentado pode ser de quatro tipos: Comercial, Informacional, Experiencial e Comportamental.

Conteúdo Comercial

Contém toda a multimédia que promova vendas de produtos/serviços ou a divulgação de novas marcas. É de bastante importância no que toca aos aspetos comerciais de uma empresa.

Conteúdo Informacional

Tem como principal objetivo apresentar dados relativos a determinado serviço e/ou produto. Serve para informar a audiência sobre o estado de determinado serviço/produto e sobre possíveis decisões a tomar (ex: ecrãs dos aeroportos onde se pode ver o estado dos voos).

Conteúdo Experiencial

É mais direcionado para audiências que estão à espera de ser atendidas numa situação e lugar específicos. É, por isso, toda a multimédia que visa tranquilizar e ajudar na espera por um serviço (ex: atendimento médico onde existe um ecrã que apresenta doenças-sintomas e resoluções).

Conteúdo Comportamental

Apresenta à audiência um serviço/produto relacionado com a tarefa que querem completar, de forma a evitar esperas. É toda a multimédia que pretende substituir/controlar o comportamento de um indivíduo perante um serviço/produto (ex: fila do Banco para consultar saldo, ecrã a apresentar a possibilidade de fazer o mesmo online).

2.1.2 *Multiplatform Digital Signage*

A um dispositivo estão associados vários tipos de ecrãs/resoluções e sistemas operativos consoante a categoria em que está inserido: *mobile*, *desktop*, *werable* ou *smart TV*. Estas diferenças entre dispositivos, apresentam um problema no que toca a exibição de conteúdos em múltiplas plataformas. Assim, um sistema multiplataforma de *Digital Signage* deve ser capaz de lidar com diferentes ecrãs/resoluções e suportar diferentes sistemas operativos, de forma a tornar a visualização de conteúdos multimédia possível em múltiplos dispositivos.

2.1.3 Importância da *Digital Signage*

Este meio de apresentação de conteúdo digital dinâmico permite a fácil transmissão de uma mensagem entre o vendedor e o comprador. Pode, por isso, ser visto como um meio de comunicação indireta entre empresa-consumidor. O crescente uso de *Digital Signage* deve-se aos seguintes fatores de sucesso:[109]

- ***Return of Investment* (ROI)**: apesar de por vezes ser necessário um elevado investimento inicial para a implementação de um sistema de *Digital Signage*, a longo prazo este tende a aumentar os lucros obtidos[50].
- **Solução para publicidade**: é possível trocar os anúncios rapidamente, bem como direcionar mais facilmente a publicidade para determinado publico alvo. Outro facto importante é a impossibilidade de "desligar"este tipo de publicidade por parte dos consumidores.

- **Conetividade entre organização e consumidor**: em comparação com os meios de publicidade tradicionais (ex: jornais), a *Digital Signage* apresenta uma capacidade superior de atrair e manter a atenção de uma audiência. Devendo-se em grande parte ao tipo de conteúdos dinâmicos que pode ser apresentado.

- **Flexibilidade**: em termos dos conteúdos que podem ser apresentados, sistemas de distribuição e diferentes ecrãs suportados, a *Digital Signage* apresenta uma maior capacidade de adaptação que os sistemas tradicionais.

- **Qualidade de conteúdos**: os conteúdos que podem ser apresentados por *Digital Signage* têm uma qualidade superior (vídeos de alta definição, imagens de alta resolução, entre outros).

- **Custos e administração**: hoje em dia, com a proliferação dos mais diversos tipos de ecrãs, os custos destes tendem a diminuir. Deste modo, é possível obter estes ecrãs/dispositivos de exibição por um preço baixo. A administração de sistemas de *Digital Signage* pode ser feita de maneira fácil e com baixos custos.

Este conjunto de fatores tenta representar de uma forma geral o valor que um sistema de *Digital Signage* pode trazer para uma empresa no que toca a comunicação com nos seus clientes.

2.2 Sistemas existentes para *Digital Signage*

Esta secção tem como objetivo apresentar alguns sistemas existentes para a distribuição e exibição de conteúdos em ecrãs digitais (*Digital Signage*). Estes sistemas vão ser apresentadas com as características que foram possíveis de determinar, uma vez que na sua maioria tratam-se de soluções não gratuitas. Também vai ser realizada uma avaliação relativamente a cada um, com o intuito de perceber se garantem um conjunto de objetivos e funcionalidades específicas.

No que toca aos sistemas de *Digital Signage* utilizados atualmente, podem ser classificados em dois principais tipos: manuais ou automáticos.

Os sistemas manuais baseiam-se na utilização de dispositivos físicos para a distribuição/exibição de conteúdos em todos os ecrãs disponíveis. Este tipo de solução implica a monitorização local de todo o processo de distribuição/exibição de um conteúdo, o que leva a um maior consumo de tempo e de recursos pessoais de uma empresa. Portanto, não faz sentido a sua utilização na era tecnológica atual, onde a maioria dos dispositivos dispõe de acesso à Internet ou rede local, a qual permite implementar um sistema de *Digial Signage* mais eficaz.

Os sistemas automáticos utilizam conexões à rede para possibilitar a distribuição/exibição de conteúdos remotamente. Estes sistemas são normalmente constituídos por dois componentes: um servidor de gestão e uma aplicação destinada unicamente à visualização de conteúdos. Baseando-se em ligações cliente-servidor, onde são trocadas todas as informações necessárias para tornar o processo de distribuição/exibição de multimédia automático e sem a necessidade de monitorização física.
A análise de sistemas existentes vai ser feita tendo em conta este tipo automático de *Digital Signage*, devido às semelhança com solução a desenvolver durante este estágio.

2.2.1 Critérios de análise para sistemas automáticos existentes

Como forma de analisar os sistemas existentes de *Digital Signage*, será feita uma descrição individual de cada um deles, seguida de uma avaliação das capacidades/funcionalidades que apresentam para a distribuição/exibição automática de conteúdos. Esta avaliação vai ser realizada de acordo com um conjunto de critérios/características provenientes das funcionalidades esperadas para o sistema a desenvolver. Posto isto, os critérios/características a verificar vão ser:

- **Suporte para plataformas *desktop***: o sistema permite exibir conteúdos nos sistemas *Windows, Mac OS* e *Linux*.
- **Suporte para plataformas *mobile***: o sistema permite exibir conteúdos nos sistemas *Android* e *iOS*.
- **Gestão de dispositivos do sistema**: um utilizador consegue visualizar, editar e adicionar dispositivos destinados à visualização de conteúdos ao sistema.
- **Atribuição de informação a dispositivos**: é possível especificar os conteúdos multimédia que vão ser reproduzido por cada dispositivo, assim como a temporização da reprodução dos mesmos.
- **Comunicação em tempo real**: alterações a *playlists* atribuídas a um dispositivo são refletidas no momento que ocorrem.
- **Operações automáticas**: os dispositivos do sistema conseguem obter automaticamente os conteúdos multimédia e agendamentos que lhes são atribuídos.

2.2.2 Sistemas existentes

NoviSign's Digital Signage System

O sistema de *Digital Signage* da *NoviSign's*[59] é oferecido num modelo de *Sofware as a service* (*SaaS*) e está dividido em duas partes: a plataforma de gestão na *Cloud*, que pode ser acedida através de um *browser*; e a aplicação destinada à exibição de conteúdos, que deve estar instalada em todos os dispositivos destinados à visualização de multimédia.

A plataforma de gestão permite ao utilizador adicionar conteúdos multimédia, criar listas de reprodução de conteúdos e gerir dispositivos ou grupos destes. Já a aplicação responsável pela exibição de conteúdos só é disponibilizada para os sistemas *Android*, *Windows* e *Chrome OS*, o que resulta num suporte deficitário em termos de diferentes dispositivos.

As operações automáticas de atribuição/exibição de conteúdos em determinado dispositivo estão condicionadas à inserção de um identificador único, gerado na página de *back-office* para cada instalação da aplicação de visualização de multimédia. Por cada instalação para uso comercial é ainda necessário o pagamento de vinte dólares.

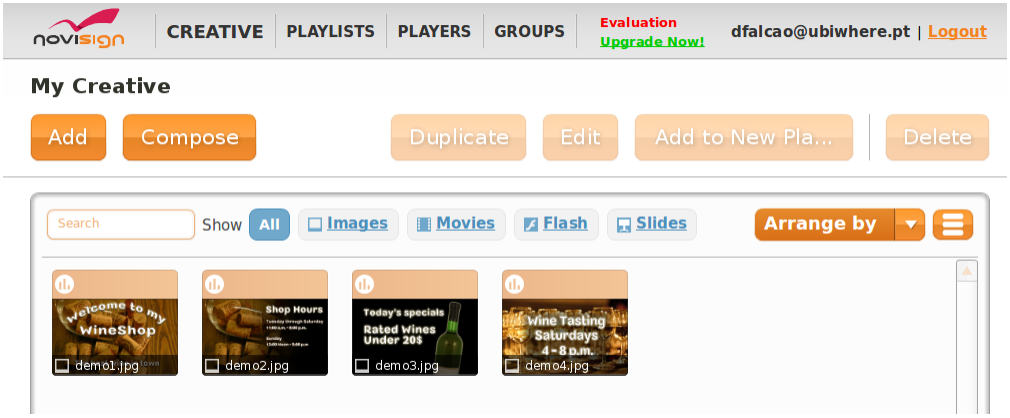


Figura 2.1: Página de *back-office* da NoviSign's.

Apesar deste sistema permitir a distribuição/exibição de conteúdos automática com configurações adicionais (inserção do identificador único), não permite a visualização destes numa grande variedade de plataformas, em concreto, não suporta os sistemas *Mac OS*, *Linux* e *iOS*. Em termos da propagação de alterações aos conteúdos a exibir por determinado dispositivo, na documentação deste sistema é dito que ocorrem no período de um minuto após serem realizadas, não podendo por isso, ser consideradas em *real time*.

A aplicação de *back-office*, onde pode ser feita a gestão do sistema, é de fácil utilização e apresenta *layouts* explícitos para o conjunto básico de funcionalidades disponíveis. Esta aplicação pode ser acedida em vários *browsers*, mediante a instalação do *plugin Adobe Flash Player*, o que pode trazer problemas nos casos em que este não é suportado.

Viewneo Digital Signage System

O sistema de *Digital Signage Viewneo*[82] permite a adição de novos dispositivos ao sistema, criação de grupos de dispositivos, criação de *playlists*/agendamento de conteúdos e ainda a distribuição automática de conteúdos. Este sistema, segue o modelo de comercialização *SaaS*, necessitando do pagamento de uma subscrição anual, para tornar possível a visualização de conteúdos nos sistemas *Android* e *iOS*.

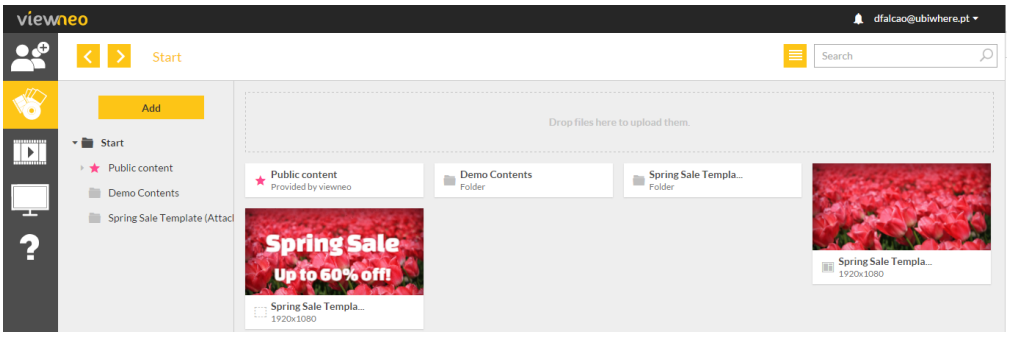


Figura 2.2: Página de *back-office* Viewneo.

A gestão do sistema pode ser feita na *Cloud*, através de uma aplicação *web* de *back-office*, acessível através de qualquer *browser* com ligação à *Internet*. Este *back-office* é de fácil utilização, contendo informação detalhada de cada funcionalidade (*pop-ups* informativos) e premiando o *drag-and-drop* de componentes. Apresenta também bastante qualidade nos *layouts* utilizados, ao promover usabilidade com a separação específica de funcionalidades baseada em grupos.

Os aspetos negativos deste sistema encontram-se na funcionalidade de gestão de dispositivos, no conjunto reduzido de plataformas suportadas e no tempo elevado de propagação de alterações. A funcionalidade de gestão de dispositivos necessita a inserção manual de um identificador único disponibilizado após a instalação da aplicação de exibição de conteúdos, para que esta possa ser reconhecida no sistema. Em termos de plataformas suportadas, é impossível exibir conteúdos nos sistemas *desktop*, *Windows, Mac OS* e *Linux*. Por fim, as alterações aos ficheiros a exibir em determinado dispositivo só são detetadas em intervalos de quinze minutos, não suportando, por isso, comunicação em tempo real.

Navori Digital Signage System

A Navori[57] oferece um sistema de *Digital Signage* numa de três formas:

- *QL Express*: versão com o menor número de funcionalidades, que utiliza um espaço dedicado na *cloud* como *back-office* do sistema.
- *QL Professional*: versão com um leque completo de funcionalidades, que também utiliza um espaço dedicado na *cloud* como *back-office*, contudo não dispõe de algumas funcionalidades de gestão.
- *QL Professional on premise*- versão mais completa do sistema, onde o *back-office* administrativo é um servidor da empresa. Esta versão, ao contrário das anteriormente descritas, oferece um conjunto mais completo de funcionalidades de gestão.

As versões do *Navori Digital Signage System* que utilizam a *cloud* como servidor de *back-office* requerem um pagamento mensal a partir de 20 dólares. Este sistema permite a distribuição automática de conteúdos, criação de *playlists* de conteúdos, criação de agendamento de reproduções e gestão de dispositivos. Possibilita

também a monitorização de informações relacionadas com a reprodução de cada *player*, criação de vários níveis de utilizadores, utilização de *data feeds* e ainda criação de *templates* de reprodução para divisão do ecrã. Todas estas funcionalidades de gestão são oferecidas por uma aplicação *web* de *back-office* denominada por *QL Manager*.

O *back-office* deste sistema apresenta grande qualidade no que toca a interface disponibilizada e permite, na mesma página, realizar todo o processo de criação de *playlists* e agendamentos de conteúdos, bem como a sua atribuição às aplicações responsáveis pela exibição. Estas aplicações são denominadas por *QL Player* e são disponibilizadas apenas para os sistemas *Android* e *Windows*.

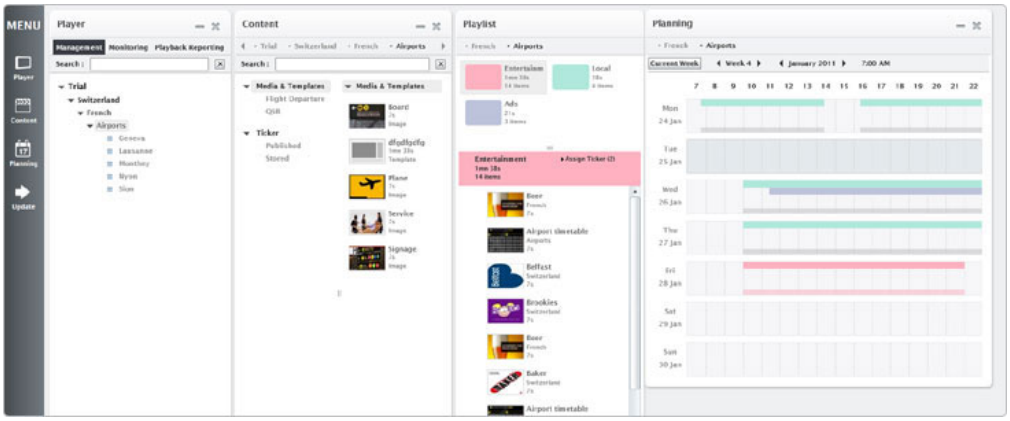


Figura 2.3: Página de *back-office* Navori.

As características distintivas deste sistema são a possibilidade de monitorização e propagação de alterações aos ficheiros a exibir em tempo real, assim como, a facilidade de utilização do sistema de agendamento de reproduções, baseado em *drag-and-drop*.

Apesar deste sistema possuir capacidades muito boas no que toca à distribuição/gestão de conteúdos, a exibição dos mesmos deixa um pouco a desejar, devido ao número reduzido de plataformas suportadas; em concreto, não suporta os sistemas *Mac OS*, *Linux* e *iOS*.

Mirabyte Digital Signage System

O sistema de *Digital Signage FrontFace*[18] oferecido pela *Mirabyte*[51] incorpora capacidades de gestão (*Display Assistant*) e exibição (*FrontFace Player*) num só *bundle* para o sistema operativo *Windows*. Com este sistema é possível exibição/distribuição automática de conteúdos, gestão de dispositivos do sistema e a criação de agendamentos de reprodução.

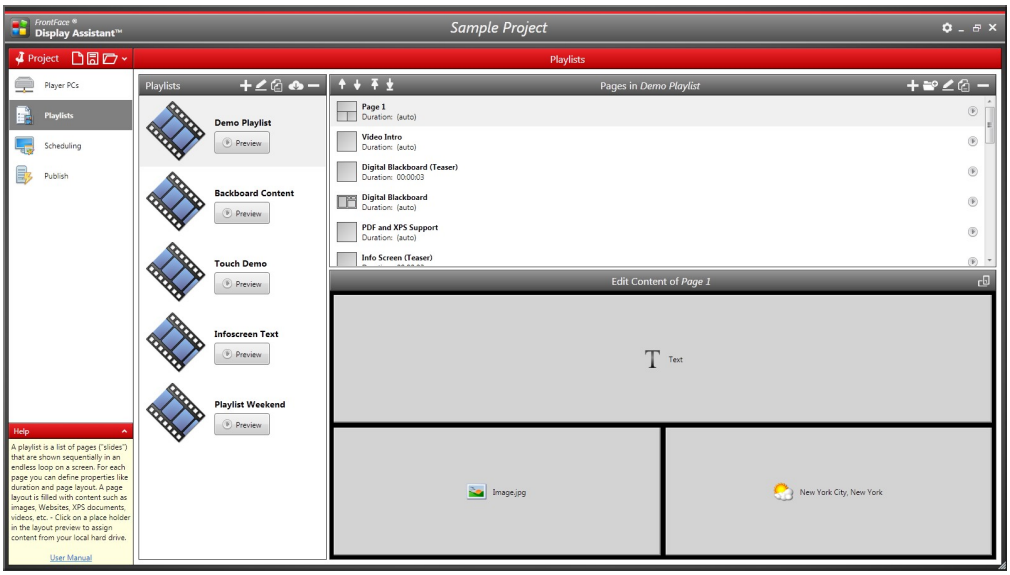


Figura 2.4: Aplicação de back-office Display Assistant.

Uma das características diferenciadoras deste sistema é a possibilidade de exibição de conteúdos *offline* a partir de dispositivos de armazenamento externos ou locais ao dispositivo. Outra característica diferenciadora é a capacidade de distribuição de alterações aos conteúdos a exibir em tempo real, utilizando uma conexão de rede.

O *back-office* deste sistema é apresentado na forma de uma ficheiro executável para o sistema operativo *Windows* e contém um conjunto de *layouts* rudimentares, com uma qualidade de gráfica muito fraca. A *interface* disponibilizada contém um conjunto elevado de elementos não explicativos da funcionalidade que despoletam o que implica a leitura do manual de utilizador para os compreender. Outro aspeto negativo, à parte do *back-office* de pouca qualidade, é a impossibilidade de gestão/exibição de conteúdos nos sistemas operativos *Android*, *iOS*, *Mac OS* e *Linux*.

UCView Digital Signage System

O sistema de *Digital Signage* da *UCView*[17] permite a distribuição automática de conteúdos, criação de listas de reprodução, registo de novos dispositivos (*players*) e criação de *layouts*/conteúdos personalizados. A reprodução de multimédia pode ser feita nos sistemas *Windows*, *Ubuntu*, *Raspberry Pi* e *Android* ou pode ser feita utilizando um dispositivo externo para o mesmo efeito. A distribuição de conteúdos (*back-office* do sistema) pode ser realizada utilizando servidores na *cloud* ou servidores locais, que devem ser obtidos para o efeito.

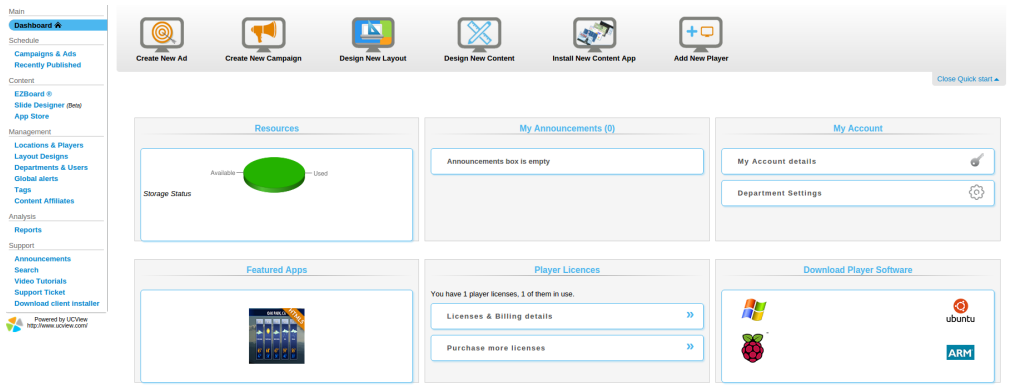


Figura 2.5: Página de *back-office* UCView.

O *back-office* de gestão deste sistema apresenta *layouts* um pouco rudimentares, mas a interface disponibilizada é de fácil utilização e intuitiva, permitindo perceber as funcionalidades disponíveis através da simples observação.

As características diferenciadoras deste sistema são: a possibilidade de ver/configurar em tempo real os dispositivos ativos no sistema e a capacidade de aceder ao histórico de operações realizada na aplicação web.

Em termos de aspetos negativos, a utilização deste sistema está condicionada pelo pagamento mensal de um plano, ao qual acresce custos por cada aplicação que se pretenda instalar, e a exibição de conteúdos não é suportada pelos sistemas *Android, iOS* e *Mac OS*.

2.2.3 Análise e comparação de sistemas existentes

Nesta secção vão ser avaliados e comparados os sistemas concorrentes apresentados na secção 2.2.2, tendo em conta o conjunto de características definidas na secção 2.2.1. Os resultados obtidos encontram-se na tabela a baixo.

●Apresenta a característica. ●Apresenta parcialmente a característica. ●Não apresenta a característica.

	Novisign's	**Viewneo**	**Navori**	**Mirabyte**	**UCView**
Suporte para plataformas desktop	●	●	●	●	●
Suporte para plataformas mobile	●	●	●	●	●
Gestão de dispositivos do sistema	●	●	●	●	●
Atribuição de informação	●	●	●	●	●
Comunicação em tempo real	●	●	●	●	●
Operações automáticas	●	●	●	●	●

Tabela 2.1: Tabela de comparação de sistemas concorrentes.

Ao analisar a tabela 2.1 e as descrições dos respetivos sistemas, é possível observar que estes apresentam um suporte deficitário de diferentes plataformas e que, na sua maioria, não suportam a comunicação de alterações dos ficheiros a exibir em tempo real.

No caso específico de uma loja de dispositivos, a utilização deste tipo de sistemas não se tornaria vantajosa, pois o número de sistemas operativos suportados é muito reduzido e, por consequente, também os dispositivos destinados à exibição de conteúdos.

Dos sistemas detalhados, os da *Novisgn's*, *Navori* e *UCView* têm um conjunto de características e funcionalidades semelhantes. Destaca-se o sistema da *Navori* que, apesar de ser pago, apresenta o melhor *back-office* (facilidade de utilização).

O sistema *Viewneo*, que apenas suporta plataformas *mobile* (*iOS* e *Android*), contém uma funcionalidade interessante onde é possível controlar o intervalo de atualização de conteúdos (de quanto em quanto tempo é que a aplicação de exibição vai buscá-los). O sistema da *Mirabyte* é o que tem menor número de plataformas suportadas, contudo é o que mostrou maior número de semelhanças à solução a desenvolver.

No caso de uma loja de retalho de dispositivos, o aspeto mais importante a ter em conta são os sistemas operativos suportados. Apesar de um sistema típico e mais geral de *Digital Signage* conseguir satisfazer as necessidades destes tipos de lojas, não é a solução mais proveitosa devido à disponibilidade de um grande número de ecrãs grátis. Assim, um sistema de *Digital Signage* deve ser personalizado/construido a pensar no caso específico da sua utilização, para se tornar realmente rentável.

2.3 Aplicações multiplataforma

Judith Bishop e Nigel Horspool descrevem uma plataforma como uma linguagem, um sistema operativo, um computador ou uma combinação de ambos [93], sendo por isso um termo que pode ter múltiplas definições. Neste relatório de estágio, o termo plataforma é utilizado tanto para referir um sistema operativo, um *hardware* específico, um navegador ou uma combinação destes, dependendo do contexto. Uma plataforma pode estar inserida dentro de uma de três categorias gerais: *desktop*, *mobile* ou *web*.

Dentro da categoria de plataformas *desktop*, o conjunto de sistemas operativos mais populares são os sistemas *Windows*, *Mac OS X* e *Linux*, utilizados nos mais diversos tipos de computadores pessoais.

A categoria de plataformas *mobile* inclui os diversos tipos de *tablet's* e *smartphone's* onde os sistemas operativos mais significantes são o *Android* e *iOS*. Estes, por sua vez, tentam oferecer o mesmo tipo de funcionalidades

das plataformas *desktop* de uma forma portável e, no momento de escrita deste relatório de estágio, são os mais utilizadas a nível global [94].

As plataformas *web* são um tipo especial, que correm em *desktop* ou *mobile*. Este tipo é utilizado de forma a aproveitar todas as capacidades da atual *World Wide Web* e os exemplos mais populares são o *Google Chrome*, o *Mozilla Firefox* e o *Internet Explorer* [5].

Posto isto, uma aplicação multiplataforma é aquela que está disponível para mais que uma das plataformas já mencionadas. Assim, como um dos objetivos do sistema a desenvolver é a exibição de conteúdos multimédia nos sistemas *Windows, Mac OS, Linux, Android* e *iOS*, é possível afirmar que durante este estágio vai ser desenvolvida uma aplicação multiplataforma.

2.3.1 Categoria de aplicações

As aplicações multiplataforma podem ser classificadas em três categorias distintas: aplicações nativas, aplicações *web* e aplicações *híbridas*.

Aplicações nativas

São normalmente produzidas com a ajuda de um *SDK* específico, que oferece o melhor acesso às capacidades nativas da plataforma. O desenvolvimento deste tipo de aplicações para diferentes plataformas implica esforços duplicados, nos quais a mesma funcionalidade tem de ser implementada em diferentes linguagens de programação, o que leva a um aumento de custos e tempo. [96] [107].

Aplicações *web*

O desenvolvimento de aplicações *web* atinge um grande número de plataformas, porque estas correm dentro de um navegador que vem incluído por defeito tanto em *smartphones* como em *pc's* (diferindo de sistema operativo para sistema operativo). As aplicações *web* são desenvolvidas normalmente usando tecnologias *Web (HTML,CSS,JS)*, as quais são compatíveis entre navegadores devido aos *standards* [75] a que estão sujeitas.

Em termos de *standards* e antes do aparecimento do *HTML5*, o *HTML* era mais utilizado para a produção e ligação de documentos estáticos (*web sites* clássicos). E para a criação de *web sites dinâmicos* era necessário o uso de outras tecnologias como: *PHP* ou *Javascript*. Este facto levou ao aparecimento do *standard HTML5*, que construiu uma tecnologia unificadora para o desenvolvimento de *web apps* (*sites* dinâmicos) [104].

Ao usar *HTML5*, um navegador é provido de capacidades das aplicações comuns (gravação de dados, trabalhar *offline*, etc), tornando-o numa plataforma iterativa para o *deploy* de aplicações e facilitando o uso de conteúdos multimédia avançados (vídeo, *svg*, etc) [101]. Ao utilizar este tipo de tecnologias é possível seguir o

paradigma *write once deploy to many*, o que permite uma maior reutilização de código.

Apesar dos grandes avanços feitos nos *standards* para a *web*, as aplicações desenvolvidas em HTML5 apresentam limitações no acesso nativo da plataforma em que estão a correr e as capacidades que podem ser acedidas (ex: microfone) necessitam de um conjunto de permissões específicas que têm de ser aceites pelo utilizador.

Aplicações híbridas

São uma categoria especial das aplicações *web* que tentam utilizar tecnologias *web* em conjunto com o acesso nativo à plataforma. Usam uma instância do *browser* desprovida de todos os controlos normais (barra de navegação, favoritos, etc) e contêm uma camada de abstração que permite o acesso a capacidades nativas da plataforma em que estão a executar.

2.3.2 Comparação e análise dos tipos de aplicações multiplataforma

Os vários tipos de aplicações multiplataforma vão ser avaliados segundo: o custo, a reutilização do código, o acesso nativo, a distribuição, a conetividade de rede, as linguagens de programação e o desempenho. Esta comparação vai permitir escolher o melhor tipo de aplicação a desenvolver para a conclusão deste projeto de estágio. Os resultados obtidos encontram-se na tabela a baixo.

	Nativa	**Híbrida**	***Web***
Custo	Alto	Baixo	Baixo
Reutilização de código	Baixo	Alto	Alto
Acesso nativo	Todas as *APIs*	Maioria das *APIs*	Poucas *APIs*
Distribuição	Lojas de aplicações	Loja de aplicações	*Internet*
Conetividade de rede	*Online* e *offline*	*Online* e *offline*	Maioritariamente *online*
Linguagens de programação	Específicas	*HTML, CSS, JS*	*HTML, CSS, JS*
Desempenho	Melhor possível	Baixo a médio	Baixo a médio

Tabela 2.2: Comparação de tipos de aplicação.

A análise dos resultados presentas na tabela anterior é a seguinte:

- **Desenvolvimento multiplataforma**:
  - As aplicações híbridas e *web* apresentam custos semelhantes (baixo), devido a uma maior reutilização de código oferecida pelas linguagens *web*.

  - As aplicações nativas são as que apresentam maior custos, pois o desenvolvimento para múltiplas plataformas implica o uso de várias linguagens de programação específicas, levando a uma menor reutilização de código entre plataformas.

- **Operações nativas/*offline***:

  - As aplicações nativas e híbridas são semelhantes em termos do acesso nativo às capacidades do dispositivo, bem como em operações *offline*.
  - Nas aplicações *web* é possível aceder a um conjunto muito limitado das capacidades do dispositivo e a capacidade de realização de operações *offline* também é muito reduzida.

- **Distribuição e desempenho**:

  - As aplicações nativas são as que apresentam melhores resultados no desempenho e na distribuição, pois utilizam *APIs* específicas. Estas aplicações podem ser distribuídas usando métodos tradicionais (instaladores) ou utilizando *App Stores*.
  - As aplicações *híbridas* e *web* têm desempenhos semelhantes, devido ao uso de tecnologias *web* com algumas diferenças técnicas no que toca à implementação (ex: camada de abstração de sistema nas aplicações híbridas), *browser* ou estado da ligação à rede. Já na distribuição, as aplicações híbridas seguem um modelo semelhante ao das nativas e as aplicações *web* só podem ser distribuídas através da *Internet* (o que por vezes pode representar um problema).

Conforme os resultados obtidos uma aplicação híbrida é a que apresenta mais vantagens para o o sistema automático de distribuição/exibição de conteúdos remoto a desenvolver.

2.4 *Frameworks* multiplataforma

Uma *framework* deve facilitar a reutilização de componentes (ex: arquitetura, módulos) e oferecer um conjunto de opções para criação rápida de aplicações [106]. Deste modo, uma *framework* multiplataforma deve permitir a reutilização de código para múltiplas plataformas, assim como oferecer *APIs* para acesso a capacidades nativas que não sejam comuns a nenhuma plataforma. No contexto do sistema a desenvolver, uma *framework* multiplataforma é um conjunto de ficheiros de código fonte, bibliotecas e ferramentas [99] que seguem os seguintes requisitos:

- Desenvolvimento para mais que uma plataforma.
- Linguagem/*API* unificadora.
- *APIs* para lidar com diferentes dispositivos.
- Ligação a serviços de *back-end*.

- Acesso às capacidades nativas da plataforma.

No que toca à construção do sistema de *Digital Signage*, e para facilitar o seu desenvolvimento, as *frameworks* devem: estar bem documentadas (documentação oficial e informações comunitárias), a sua utilização não deve acrescer custos (fator não eliminatório), facilitar a implementação dos requisitos chave (ex: exibição de vários tipos de conteúdos multimédia) e possibilitar o uso futuro da mesma para outros projetos de implementação multiplataforma.

2.4.1 Análise de *frameworks*

Esta secção tem como o objetivo apresentar as *frameworks* consideradas para o desenvolvimento do sistema proposto. As *frameworks* aqui descritas são o resultado da pesquisa efetuada sobre tecnologias de desenvolvimento que podem ser utilizadas para a implementação das aplicações multiplataforma. A partir do conjunto de *frameworks* consideradas, foram escolhidas aquelas que efetivamente vão ser utilizadas.

As *frameworks* vão ser analisadas tendo em conta o seu tipo e segmento. Assim, as *frameworks* vão estar divididas em quatro grupos fundamentais: baseada em *browser container*, base em *drag-and-drop*, baseada em máquinas virtuais e baseadas em *cross-compiling*. Dentro deste grupos cada *framework* vai estar classificada segundo o segmento que tenta atingir: *desktop* ou *mobile*.

As características diferenciadores que vão influenciar a escolha das *frameworks* a utilizar dividem-se em fatores subjetivos (aprendizagem/qualidade da *framework*) e fatores objetivos (relacionados com implementação).

- **Aspetos subjetivos:**
  - **Qualidade/quantidade de informação**: visa mitigar problemas futuros de utilização da *framework* em termos de instalação, configuração, manuseamento e resolução de problemas (relacionados com implementação), com base na documentação oficial e comunitária disponível.
  - **Curva de aprendizagem**: permite classificar a facilidade de utilização de uma *framework*, tendo em conta os conhecimentos específicos (linguagens de programação, processos de engenharia, entre outros), que o estagiário tinha na data de escrita deste relatório.
  - **Longevidade**: pretende representar a estabilidade e probabilidade da *framework* sobreviver. Um projeto que está a começar está pouco maduro, sendo por isso, menos estável e mais propício ao aparecimento de *bugs*[100]. Enquanto num projeto mais antigo é menor a possibilidade de este ser descontinuado e maior a sua estabilidade.[98].

- **Backer**: é importante saber qual a equipa que mantém determinada *framework*, pois se esta pertencer a uma empresa/companhia, a sua qualidade pode ser comprovada com base em outros produtos desenvolvidos.

- **Aspetos objetivos:**

  - **Plataformas suportadas**: a *framework* suporta os sistemas operativos de *deploy* necessários (dentro do respetivo segmento). Neste aspeto, também foram consideradas plataformas adicionais suportadas, de forma a tornar possível um trabalho futuro de expansão do sistema.
  - **Operações *offline***: a *framework* possibilita/facilita operações *offline*, como a leitura e gravação de ficheiros locais.
  - **Reprodução de vários formatos multimédia**: como uma das principais funcionalidades do sistema a desenvolver é a exibição de vários tipos de conteúdos multimédia, a *framework* deve possibilitar o uso de diferentes formatos de ficheiros (ex: SVG, MP4, entre outros).
  - **Comunicação com servidores externos de *back-end***: a *framework* permite uma fácil integração com servidores externos.
  - **Preço de utilização**: a *framework* não deve ter custos de utilização inerentes, para tornar possível a conclusão do projeto sem custos adicionais.

Frameworks baseadas em *browser containers*

Um *browser container* é normalmente desprovido de todo o tipo de controlos encontrados num *browser* comum, servindo só o propósito de interpretar a aplicação subjacente escrita usando tecnologias *web*. Este tipo de *frameworks* visa facilitar a criação de aplicações *web* dedicadas a este género especial de *browser*.

Electron (*desktop*)

O *Electron*[23] é uma *framework open-source* disponibilizada e mantida pelo *Github* deste 2014, que permite o desenvolvimento de aplicações com capacidades nativas para sistemas operativos *desktop*. Trata-se de uma *framework* capaz de produzir aplicações com aspeto e funcionalidades nativas, utilizando *Javascript, CSS e HTML*.

Algumas das características mais importantes desta *framework*, são:

- Suporte para os sistemas operativos *Mac OS* , *Linux* e *Windows*.
- Possibilidade de utilizar pacotes do *Node.js*[58] e pacotes externos disponibilizados pelo *Node package manager* (npm)[60]. Dentro deste conjunto de pacotes, é possível:

  - A criação de ficheiros executáveis - (*electron-packager*)[25].
  - A criação de instaladores - (*electron-builder*)[24].
  - A criação de aplicações com a capacidade de *updates* automáticos - (*electron-updater*)[28].
  - O manuseamento de ficheiros locais, através de um pacote denominado por fs.[31].
- Disponibilização de um conjunto de opções específicas, para dotar as aplicações criadas com um aspeto nativo.
- Conjunto de aplicações criadas utilizando esta *framework*, das quais se destacam:
  - A aplicação *desktop* do *Slack*[72], que permite uma comunicação fácil entre equipas.
  - *Visual Studio Code*[85], um *IDE* da *Microsoft* para o desenvolvimento rápido de aplicações *web*.
  - E uma aplicação para o *streaming* de vídeos denominada por *Streamio*[76].

NW.js (*desktop*)

NW.js[62], conhecido anteriormente por *node-webkit*, é uma *framework open-souce* que existe desde 2011. Esta *framework* é mantida pela *Intel* e permite o desenvolvimento de aplicações *desktop* multiplataforma. Semelhantemente à *framework* apresentada no ponto anterior, esta é apta para produzir aplicações com capacidades nativas utilizando tecnologias como *Javascript, HTML e CSS*.

As características a destacar são:

- O suporte para os sistemas operativos *Mac OS*, *Linux* e *Windows*.
- A possibilidade de utilizar as capacidades do *Node.js*[58], assim como pacotes externos disponibilizados pelo *npm*[60]. Realçando-se pacotes como:
  - O *nw-builder*[63], que permite fazer o *build* automático de aplicações criadas nesta plataforma.
  - O pacote *nw-test-runner*[81], que disponibiliza uma maneira simples de fazer testes unitários a aplicações criadas em *NW.js*.
  - *nw-updater*[61], que permite a adição de *updates* automáticos em *background*.
- A possibilidade de acesso a capacidades nativas como:
  - *Clipboard*[9], viabilizando o acesso à área de cópia dos sistemas operativos suportados.
  - *Menu*[48], proporcionando a construção das barras de menus encontradas em aplicações nativas.
  - *Shorcut*[70], permitindo o acesso a determinada funcionalidade premindo uma combinação de teclas.

Apache Cordova (*mobile*)

O *Apache Cordova*[4] é uma *framework open-source* que se originou da versão proprietária *PhoneGap*, doada pela *Adobe* à *Apache*. É mantida pela *Apache Software Foundation* (ASF)[89] e permite o desenvolvimento de aplicações *mobile* multiplataforma. Ao usar tecnologias como *HTML, CSS* e *Javascript* é possível desenvolver aplicações para sistemas como *Windows Phone*, *Android*, *iOS*, entre outros.

Algumas das características a destacar são:

- O uso de componentes totalmente nativos em conjunto com componentes multiplataforma.
- A utilização de *plugins*, para abstrair o acesso a capacidades nativas de múltiplas plataformas. Realçando-se os *plugins*:
  - O *cordova-plugin-file*[30] permite escrita/leitura de ficheiros utilizando uma API multiplataforma.
  - O *Cordova-plugin-file-tranfer*[32] é um *plugin* que facilita o upload/download de ficheiros externos ao dispositivo.
  - O *Plugin cordova-plugin-device*[16] favorece o acesso às informações do dispositivo em que a aplicação está a executar.
- A possibilidade de empregar uma "aplicação de linha de comandos"[8] para gestão de um projeto criado nesta *framework*. As capacidades de gestão incluem o adicionar/remover de plataformas a suportar pela aplicação final, adicionar/remover *plugins* para uso durante o desenvolvimento, entre outros.

Frameworks com base em *drag-and-drop*

Este tipo de *frameworks* é baseado na noção de *drag and drop* de componentes pré feitos que podem ser personalizados. Dispõem de um *IDE* com capacidades de *highlight* de linguagem, opções de *debug* e compilação de código nativo para diversas plataformas.

Xojo (*desktop* e *mobile*)

Xojo[55] é uma *framework* que permite o desenvolvimento multiplataforma, utilizando componentes *drag-and-drop* pré-definidos, em conjunto com uma linguagem de alto nível orientada para objetos baseada em *Visual Basic*, denominada por *Xojo Language*[7]. Proporciona o desenvolvimento de aplicações para os sistemas *desktop Windows*, *Mac OS* e *Linux* e para o sistema *mobile iOS*.

Algumas das características a destacar desta *framework* são:

- A disponibilização de um *IDE* próprio para o desenvolvimento de aplicações.

- A possibilidade de desenvolver aplicações *web* e também aplicações direcionadas para a consola.

- A documentação oficial, onde são cobertos todos os aspetos do seu uso.

- O conjunto de casos de estudo de aplicações desenvolvidas com recurso a esta *framework*.

- A possibilidade de utilização de *third party add-ons* para extensão das suas funcionalidades.

Esta *framework* pode ser utilizada de forma gratuita para desenvolver aplicações de teste, sendo necessário o pagamento de uma subscrição para fazer o *deploy* de aplicações terminadas. O montante a pagar por uma subscrição aumenta conforme o número de plataformas e capacidades adicionais a suportar, pelo que a utilização desta *framework* traria custos adicionais para o desenvolvimento do projeto.

***EachScape* (*mobile*)**

EachScape[34] é uma *framework* que permite o desenvolvimento de aplicações para os sistemas operativos mobile *Android* e *iOS*, utilizando componentes *drag-and-drop* em conjunto com componentes nativos.

Algumas das características a destacar são:

- A disponibilização de um *IDE* na *cloud* que proporciona o desenvolvimento *online* de aplicações.
- A disponibilização de um conjunto de componentes específicos para a criação de *user interfaces* (ui).
- A possibilidade de gerar código fonte nativo para múltiplas plataforma de uma só vez.
- A possibilidade de conexão a bases de dados externas à aplicação.
- A *framework* segue o padrão *MVC*[54].

A publicação de aplicações desenvolvidas nestas *framework* é condicionada pelo pagamento de uma subscrição. Ao pagar esta taxa é possível utilizar funcionalidades como adição de anúncios, adição de componentes de análise de tráfego e ainda a utilização de um sistema de pagamentos.

***LiveCode* (*desktop* e *mobile*)**

LiveCode[45] é uma *framework* que permite o desenvolvimento de aplicações multiplataforma para *desktop* e *mobile*, em concreto, para os sistemas *Windows*, *Linux*, e *Android*. Possibilita o rápido desenvolvimento de UI's, utilizando o *drag-and-drop* de componentes pré-definidos.

Algumas das características a destacar são:

- A disponibilização de um IDE próprio onde é facilitado o seu uso.
- A utilização de uma linguagem de alto nível, sem a necessidade de um compilador.
- O desenvolvimento de aplicações para múltiplas plataformas a partir de um ficheiro de código fonte único (*single code base*).
- A disponibilização de um conjunto de serviços pagos, que visam ajudar no processo de desenvolvimento de *software*.
- A criação de aplicações pode ser feita em sistemas conhecidos como *Windows* e *Linux*.
- A disponibilização de uma loja de *addons*.

Esta *framework* pode ser utilizada sem custos adicionais para o desenvolvimento de aplicações *open-source*, tornando mais difícil a sua comercialização. Para o desenvolvimento de aplicações *closed-source*, mais indicadas para o comércio, é necessário o pagamento de uma licença anual, que permite aceder a capacidades adicionais como proteção do código fonte e acesso ao fórum de suporte.

***Frameworks* baseadas em máquinas virtuais**

Este tipo de *frameworks* visa a criação de aplicações para uma determinada máquina virtual. Estas aplicações por sua vez, vão estar disponíveis em todas as plataformas com suporte para a máquina virtual usada.

***Xamarin* (*mobile*)**

Xamarin[52] é uma *framework* destinada ao desenvolvimento de aplicações *mobile* multiplataforma. Esta suporta a utilização da linguagem C# para a produção de aplicações, que correm nas plataformas *iOS*, *Android* e *Windows Phone*; e utiliza um *.NET runtime*, denominado por *Mono*[35], para potenciar as suas capacidades de *cross-platform*.

O conjunto de características a destacar é o seguinte:

- A possibilidade de utilização de um de dois IDE'S (*Visual Studio*[84] ou *Xamarin Studio*[3]).
- A possibilidade de desenvolver aplicações para *werables* e *smartTV's*.
- A possibilidade de construção de *UIs* nativas para múltiplas plataformas, usando um *single code base* (*Xamarin Forms*[6]).
- A possibilidade de utilização de todas as capacidades nativas dos sistemas operativos suportados.
- A disponibilização de uma loja de componentes, que permite estender as funcionalidades das aplicações desenvolvidas.

- A possibilidade de geração de ficheiros executáveis.

O uso desta *framework* para o desenvolvimento do projeto proposto implicaria o uso de uma linguagem de programação com que o estagiário não se sente à vontade, o que representa um aspeto negativo.

Codename One (*desktop* e *mobile*)

Codename One[12] é uma framework para o desenvolvimento multiplataforma *desktop* e *mobile*, que permite a construção de aplicações para os sistemas *Windows*, *Mac OS*, *iOS* e *Android* utilizando a linguagem de programação *Java*.

As características a destacar são:

- A possibilidade de utilização de vários *IDEs* (*Eclipse*[21], *NetBeans*[88] ou *IntelliJ/IDEA*[39]).
- O acesso a todas as capacidades nativas dos sistemas suportados.
- A geração de código totalmente nativo, para as plataformas suportadas.
- A possibilidade de construção de interfaces gráficas utilizando componentes *drag-and-drop*.
- A possibilidade de gerar aplicações *Javascript* direcionadas para *browsers*.

Para gerar aplicações multiplataforma, esta *framework* utiliza *build servers* na *cloud*, os quais devolvem ficheiros binários nativos que podem ser executados nas plataformas suportadas. A versão não paga possibilita o desenvolvimento de apenas uma aplicação para os sistemas *iOS* e *Android*. Com o pagamento mensal de uma subscrição, é possível gerar qualquer número de aplicações para os sistemas *desktop Windows* e *Mac OS* e para os sistemas *mobile iOS* e *Android*. É ainda possível o uso de *push notifications*, ferramentas de testes unitários, entre outros.

Frameworks baseadas em *cross-compiling* (*binary files*)

Este tipo de *frameworks* baseia-se na compilação *AOT (Ahead-of-time)*, criando executáveis otimizados para diferentes plataformas. Normalmente usam linguagens como C, C++ ou *Objective* C e fazem uso de uma *Bridging Layer* para eliminar a necessidade de escrita de código nativo a cada plataforma.

8th (*desktop* e *mobile*)

8th[2] é uma *framework* que permite o desenvolvimento multiplataforma para os sistemas *Android*, *Windows* e *Linux*. Segue o paradigma *write once deploy everywhere*, onde com um código fonte único é possível gerar uma aplicação para as todas as plataformas suportadas.

As características a destacar são:

- A possibilidade de encriptação de código fonte.
- A capacidade de criação de uma UI única para todas as plataformas.
- A compilação de aplicações para código nativo das plataformas suportadas.
- O suporte para acesso a base de dados como *SQLite*[74].
- O suporte para utilização de bibliotecas externas.
- A disponibilização de um IDE dedicado ao seu uso.

Na versão grátis desta *framework*, as aplicações geradas só podem ser utilizadas durante um pequeno período de tempo (3 dias). Para o desenvolvimento de aplicações sem um tempo limite de vida é necessário o pagamento de uma taxa mensal ou anual, sendo que, este pagamento permite também o acesso a mais bibliotecas, suporte ao desenvolvimento adicional e acesso a funcionalidades ainda não divulgadas publicamente.

NativeScript (*mobile*)

NativeScript[13] é uma *framework* para desenvolvimento multiplataforma mantida pela *Telerik*[77], que permite a construção de aplicações para os sistemas *iOS* e *Android*. É possível a utilização da linguagem *Javascript* a partir de um ficheiro de código fonte único, para atingir todos os sistemas suportados.

Algumas das características a destacar são:

- As aplicações criadas são totalmente nativas.
- A possibilidade de acesso a todas as *APIs* nativas dos sistemas operativos suportados.
- A possibilidade de construção de *UIs* totalmente nativas.
- A utilização do sistema de pacotes *npm*[60], ou *CocoaPods*[10] para a adição de *plugins* externos à *framework*.
- A disponibilização de uma extensão para o *Visual Studio Code*[85].
- A Possibilidade de utilização da *framework Angular* 2[64], permitindo a reutilização de código *web* para a construção de aplicações *mobile* nativas.

Esta *framework* apresenta um *tutorial* introdutório onde são cobertas, de uma forma simples e intuitiva, as funcionalidades disponíveis. Este conjunto de passos permite que um utilizador não familiarizado com a *framework* consiga começar a utilizá-la sem a necessidade de conhecimentos específicos, e ainda uma aprendizagem baseada no conceito de *"hands on"*.

2.4.2 Comparação de *frameworks* e conclusões

Tendo em conta as *frameworks* apresentadas, e de acordo com as suas capacidades, é possível utilizar uma combinação destas para a construção do sistema pretendido. Assim, como forma de avaliar as opções que mais se adequam às necessidades e objetivos, as *frameworks* (tecnologias) vão ser comparadas. Esta comparação vai ser feita segundo o conjunto de características apresentadas na Secção 2.4.1, de forma a que a posterior escolha das *frameworks* apresente o mínimo de barreiras e desvantagens em termos de implementação.

Na análise individual das *frameworks* apresentada anteriormente, é possível verificar que as *frameworks Xojo*, *LiveCode*, *Codename One* e *8th* permitem o desenvolvimento para o segmento *desktop* e *mobile*; as *frameworks Electron* e *Nw.js* para o segmento *desktop* e as *frameworks Apache Cordova*, *EachScape*, *Xamarin* e *NativeScript* para o segmento *mobile*.

Como existe um conjunto de *frameworks* que permite o desenvolvimento tanto para sistemas operativos *mobile* como *desktop*, estas vão ser o primeiro alvo de comparação, por forma a perceber se a implementação da aplicação de exibição de conteúdos nos sistemas *Windows*, *Linux*, *Mac OS*, *Android* e *iOS* é possível através do uso de apenas uma tecnologia.

Característica	***Xojo***	***LiveCode***	***Codename One***	***8th***
Qualidade e quantidade de informação	Alta	Alta	Média	Baixa
Curva de aprendizagem	Alta	Média	Média	Alta
Longevidade	2013	2010	2012	2013
Backer	*Xojo, Inc.*	*LiveCode, Ltd*	*Codename One, Ltd*	*Aaron High-Tech, Ltd*
Plataformas suportadas	*Windows*, *Mac OS*, *Linux* e *iOS*	*Windows*, *Linux* e *Android*	*Windows*, *Mac OS*, *iOS* e *Android*	*Windows*, *Linux* e *Android*
Operações *offline*	Suporta	Suporta	Suporta	?????
Formatos suportados	Suporta	Suporta	Suporta	Suporta
Comunicação externa	Suporta	Suporta	Suporta	Suporta
Preço de utilização	Com custos	Com custos	Com custos	Com custos
Pontuação	8	9	8	4

Legenda
2-Pontos
1-Ponto
0-Pontos

Tabela 2.3: Comparação de *frameworks* que permitem o desenvolvimento para *desktop* e *mobile*.

Na Tabela 2.3 é apresentada a comparação de *frameworks* que possibilitam o desenvolvimento de aplicações para sistemas operativos *mobile* e *desktop*. Relativamente a essas *frameworks*, é possível verificar que todas elas apresentam custos de utilização e que nenhuma apresenta suporte para todas as plataformas requeridas pela aplicação de exibição de conteúdos. Conclui-se, portanto, que para atingir todos os sistemas *desktop* e *mobile* necessários têm de ser utilizadas pelo menos duas *frameworks* de desenvolvimento multiplataforma.

Apesar de poderem ser utilizadas duas *frameworks* da tabela anterior para suportar todos os sistemas necessários, a compatibilidade destas deixa um pouco a desejar. Na medida que, *Xojo* e *8th* utilizam linguagens de programação próprias, *LiveCode* é baseada no *drag-and-drop* de componentes e *Codename One* utiliza a linguagem de programação *JAVA*. Estas diferenças impossibilitam reutilização de código, dando origem a esforços duplicados, necessidade de aprendizagem de duas tecnologias completamente diferentes e aumento do tempo de desenvolvimento. Com isto em mente, as *frameworks* dedicadas aos segmentos *mobile* e *desktop*, respetivamente, vão ser comparadas de forma a perceber se é possível usar uma combinação destas mais proveitosa para suportar os sistemas requeridos.

Legenda:	2-Pontos	1-Ponto	0-Pontos

	Desktop		***Mobile***			
Característica	***Electron***	***NW.js***	***Cordova***	***EachScape***	***Xamarin***	***NativeScript***
Qualidade e quantidade de informação	Alta	Alta	Alta	Média	Alta	Alta
Curva de aprendizagem	Baixa	Baixa	Baixa	Média	Alta	Baixa
Longevidade	2013	2011	2011	????	2011	2014
Backer	***GitHub***	***Intel***	***Apache***		***Microsoft***	*Telerik AD*
Plataformas suportadas	*Mac OS, Linux e Windows*	*Mac OS, Linux e Windows*	*Android e iOS*	*Android e iOS*	*Android e iOS*	*Android e iOS*
Operações *offline*	Suporta	Suporta	Suporta	Difícil suporte	Suporta	Suporta
Formatos suportados	Suporta	Suporta	Suporta	??????	Suporta	Suporta
Comunicação externa	Suporta	Suporta	Suporta	Suporta	Suporta	Suporta
Preço de utilização	Grátis	Grátis	Grátis	Com custos	Com custos	Grátis
Pontuação	14	14	14	7	10	14

Tabela 2.4: Comparação de *frameworks* que possibilitam o desenvolvimento para *desktop* ou *mobile*.

Na Tabela 2.4 é possível observar que todas as *frameworks* de desenvolvimento para o segmento *desktop* suportam as plataformas *Mac OS*, *Linux* e *Windows*, requeridas pela aplicação de exibição de conteúdos; assim como, todas as do segmento *mobile* suportam os sistemas *Android* e *iOS*. Posto isto, é possível a utilização de uma das *frameworks* dedicadas a *desktop* e outra a *mobile* como ferramentas de desenvolvimento para este projeto de estágio. Contudo, para o segmento *mobile*, as *frameworks EachScape* e *Xamarin* apresentam menores pontuações que *Cordova* ou *NativeScript*, sendo por isso desconsideradas.

As restantes *frameworks* (*Electron*, *Nw.js*, *Cordova* e *NativeScript*) utilizam linguagens de programação para *web*, *HTML*, *CSS* e *Javascript*, o que possibilita uma maior reutilização de código na utilização de um conjunto de duas destas *frameworks*, em relação às da Tabela 2.3.

Como existem duas *frameworks* para cada segmento com pontuações iguais, vai ser necessário considerar características adicionais para a escolha das tecnologias a usar. Posto isto, as características para desempate são:

- **Aplicações populares já desenvolvidas usando a *framework* e utilizadores significativos (empresas)**: ao analisar aplicações já desenvolvidas e seus *developers*, é possível explorar as capacidades "reais"oferecidas pela *framework*, bem como comprovar a sua qualidade através do valor dos seus utilizadores (empresas/aplicações).
- **Tecnologias semelhantes usadas na *Ubiwhere***: a análise de tecnologias semelhantes utilizadas na empresa possibilita a reutilização de conhecimentos para o uso futuro da *framework*.

O *Electron* e o *NW.js* são duas *frameworks* com capacidades semelhantes. Apresentam uma lista grande e bastante completa de aplicações já desenvolvidas. O *Electron*, apesar de ser uma *framework* menos madura que o *NW.js*, apresenta uma lista de aplicações muito maior que a anterior. Esta lista é composta por aplicações populares como: *Atom, Slack, Visual Studio Code, Facebook Messenger*, entre outras; as quais foram desenvolvidas pelas empresas *GitHub*, *Slack Technologies* ,*Microsoft* e *Facebook*, respetivamente. Já para aplicações implementadas utilizando *NW.js*, destaca-se o *Intel XDK*, desenvolvido pela Intel.

Concluindo, as duas *frameworks* foram utilizadas para a construção de aplicações populares por parte de grandes empresas, nas quais podem ser encontradas algumas funcionalidades da aplicação *desktop* multiplataforma a desenvolver. No entanto, o *Electron* apresenta um maior número de aplicações populares e um maior reconhecimento na comunidade *open-source*, devido à empresa que mantém esta *framework* (*GitHub*).

Cordova e *NativeScript* são duas *frameworks* com capacidades semelhantes na construção de aplicações multiplataforma para o segmento *mobile*. Em termos de aplicações já desenvolvidas, não há nenhuma que se destaque e as aplicações encontradas são na sua maioria criadas por empresas pequenas. Um aspeto im-

portante a realçar na comparação destas *frameworks* é a utilização do *Cordova* na plataforma *Monaca* e de *NativeScript* ser utilizado na plataforma Telerik. Outro ponto que favorece a utilização do *Cordova* é o suporte para sistemas operativos *mobile* adicionais como o *FireOS* ou *BlackBerryOS*, permitindo a eventual expansão do sistema a desenvolver.

Posto isto, na tabela abaixo é apresentada a comparação final de *frameworks*, segundo os critérios de desempate já referidos.

	Desktop		***Mobile***	
Caracteristica	**Electron**	**Nw.js**	**Cordova**	**NativeScript**
Aplicações populares				
Utilizadores significativos				
Usado na Ubiwhere			Ionic	
Pontuação	4	3	4	2
Pontuação Anterior	14	14	14	14
Pontuação Final	18	17	18	16

Tabela 2.5: Comparação final de *frameworks*.

Legenda:	2-Pontos	1-Ponto	0-Pontos

Na Tabela 2.5 é possível verificar que a *framework Electron* é a que apresenta melhores resultados para o segmento *desktop*, sendo que na *Ubiwhere* não são utilizadas nenhumas tecnologias semelhantes. Já para o segmento *mobile*, é possível observar que a *framework Cordova* é a que apresenta maior pontuação. Isto deve-se ao facto de que na *Ubiwhere* é utilizada uma ferramenta dedicada ao desenvolvimento de *UIs* para *mobile* (Ionic) que utiliza *Cordova* para suportar diferentes sistemas operativos *mobile*.

Para concluir, é possível afirmar que a utilização de *Electron* em conjunto com *Cordova* é a opção que vai trazer menos desvantagens e entraves à implementação da aplicação de exibição de conteúdos para os sistemas *Windows*, *Mac OS*, *Linux*, *Android* e *iOS*. Pois estas *frameworks* vão facilitar a reutilização de código (as duas utilizam as mesmas linguagens de programação), são de utilização grátis e a quantidade e qualidade da informação encontrada sobre estas é muito boa, entre outros (ver Tabela 2.4 e Tabela 2.5), o que se reflete nas suas pontuações finais. Posto isto, durante este estágio vão ser implementadas duas aplicações multiplataforma: uma para sistemas operativos *desktop* utilizando *Electron*, e outra para sistemas operativos *mobile*, utilizando *Cordova*.

Capítulo 3

Metodologia de Trabalho e Planeamento

Este capítulo visa apresentar o planeamento do projeto ao longo dos dois semestre, assim como os riscos, metodologias e ferramentas associadas ao mesmo.

3.1 Metodologia e Ferramentas utilizadas

Para metodologia de desenvolvimento foi escolhida a já utilizada pela empresa, denominada *Scrum*[110]. Esta permite o desenvolvimento do trabalho faseado e baseado em *sprints*.

De forma a gerir as diferentes tarefas associadas ao trabalho, será usado o *Redmine*[65] em conjunto com reuniões periódicas de acompanhamento do projeto. O uso do *Redmine* por parte da *Ubiwhere* está inserido na sua certificação em *Capability Maturity Model* (CMMI)[102], servindo para reunir métricas dos variados processos da empresa. O CMMI[102] é um modelo de maturidade desenvolvido pela *Software Engineering Institute* (SEI) da Universidade Carnegie Mellon e contém um conjunto de boas práticas que visam o melhorar o processo de desenvolvimento de um produto, desde a sua conceção até à sua entrega e manutenção. Estas boas práticas ao serem seguidas tendem a aumentar a qualidade do produto final[102].

As *sprints* de desenvolvimento de código vão ter a duração de duas semanas e no final de cada uma é realizada uma reunião de projeto com o objetivo de avaliar o que foi elaborado. Em cada *sprint* vai ser implementado um conjunto de requisitos funcionais, apresentados no Apêndice A na forma de *User Stories*, tendo em conta a sua prioridade.

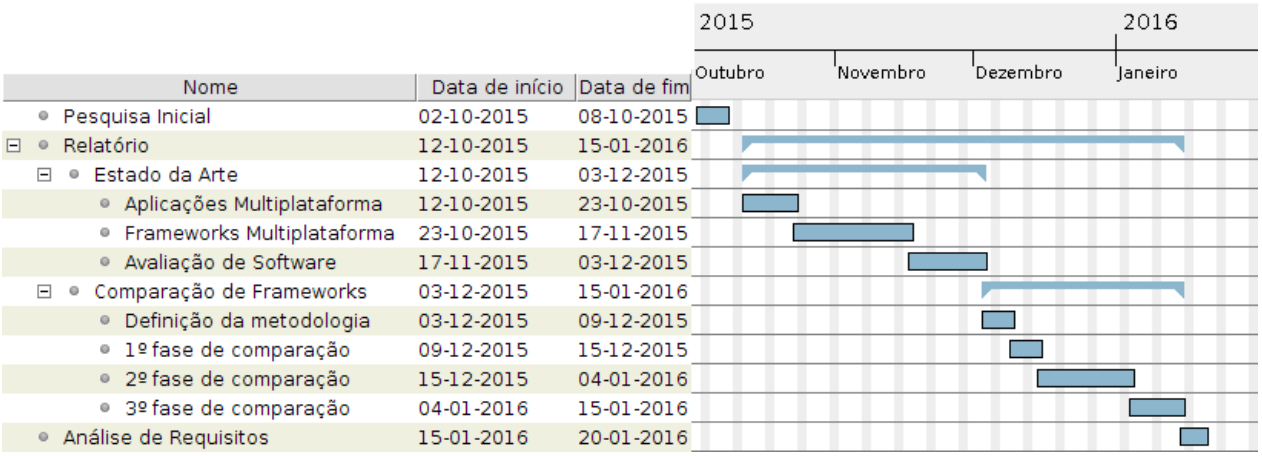
Em termos de ferramentas de apoio: é utilizado o *GitLab*[11], como forma de alojamento e controlo de versões

de código; o *Redmine*[65], para a gestão de tarefas e registo das horas despendidas; o *Slack*[72], como forma de comunicação interna; e o *WebStorm*[87] como *IDE* de programação.

3.2 Planeamento

Nesta secção será apresentado o planeamento do trabalho a desenvolver e já desenvolvido ao longo dos dois semestres que constituem o estágio.

3.2.1 Planeamento 1º Semestre



Nome	Data de início	Data de fim
Pesquisa Inicial	02-10-2015	08-10-2015
Relatório	12-10-2015	15-01-2016
Estado da Arte	12-10-2015	03-12-2015
Aplicações Multiplataforma	12-10-2015	23-10-2015
Frameworks Multiplataforma	23-10-2015	17-11-2015
Avaliação de Software	17-11-2015	03-12-2015
Comparação de Frameworks	03-12-2015	15-01-2016
Definição da metodologia	03-12-2015	09-12-2015
1ª fase de comparação	09-12-2015	15-12-2015
2ª fase de comparação	15-12-2015	04-01-2016
3ª fase de comparação	04-01-2016	15-01-2016
Análise de Requisitos	15-01-2016	20-01-2016

Figura 3.1: Tempo despendido no primeiro semestre.

O *Gantt* na Figura 3.1 representa o tempo despendido por tarefa ao longo do primeiro semestre. As tarefas presentes neste *Gantt* são maioritariamente de pesquisa sobre o tema e foco inicial do estágio, que era um estudo aprofundado de *frameworks* multiplataforma. Mostra as fases teóricas efetuadas, para possibilitar uma comparação científica e detalhada das várias *frameworks* encontradas que potencializavam o desenvolvimento de aplicações para mais que uma plataforma.

3.2.2 Desvios ao planeamento do 1º Semestre

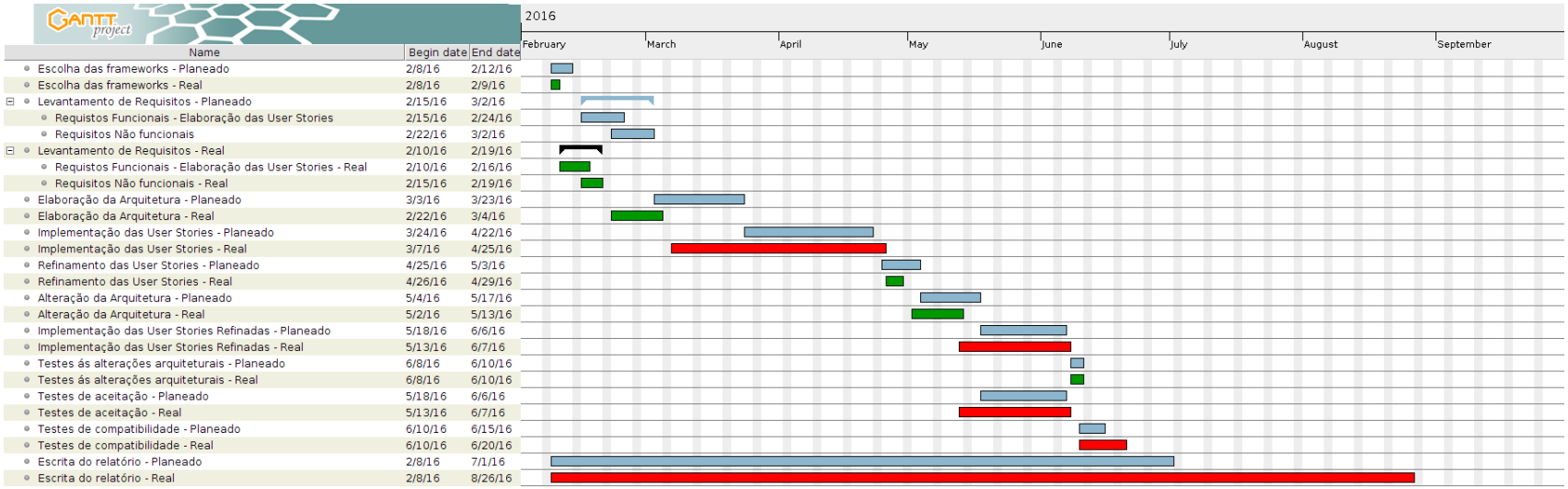
No primeiro semestre, houve uma reformulação dos objetivos do estágio. Inicialmente contavam com o estudo aprofundado de *frameworks* para desenvolvimento multiplataforma e com a implementação de uma prova de conceito que suporta-se a execução em vários sistemas operativos. No estudo aprofundado de *frameworks* foram encontrados diversos problemas em termos da comparação/avaliação da qualidade destas. Estes problemas deveram-se ao facto de o tema em foco ser completamente novo para o estagiário e das tecnologias a comparar conterem características únicas que dificultavam uma avaliação científica. Isto, em

conjunto com o número elevado de cadeiras a realizar no primeiro semestre, resultou em tempo insuficiente para realizar a comparação de *frameworks* com a qualidade esperada, originando a já referida reformulação de objetivos de estágio. O objetivo principal passou a ser o desenvolvimento de uma ou mais aplicações utilizando *frameworks* multipaltaforma e a integração destas com um componente já criado.

3.2.3 Planeamento 2º Semestre

Nesta secção é apresentado o planeamento e as tarefas realizadas no 2º Semestre. Na Figura 3.2 é possível observar o diagrama de *Gantt* que contém o tempo despendido em cada tarefa considerada relevante. Ainda neste diagrama, são apresentados os desvios ao planeamento inicial. A descrição das tarefas a realizar é a seguinte:

1. **Escolha das *frameworks* a utilizar**: para esta escolha vai-se recorrer ao estudo de algumas características já realizado.
2. **Requisitos funcionais - Elaboração das *User Stories***: levantamento e documentação na forma de *US* dos requisitos funcionais do sistema.
3. **Requisitos não funcionais**: elaboração e documentação dos atributos de qualidade que o sistema deve ser capaz de garantir.
4. **Elaboração da Arquitetura**: definição e estruturação da arquitetura para o sistema.
5. **Implementação das *User Stories***: construção das funcionalidades do sistema com maior prioridade.
6. **Refinamento das *User Stories***: reestruturação e alteração das *User Stories* do sistema para acomodar as novas necessidades do cliente (Ubiwhere).
7. **Alteração da Arquitetura**: introdução de mudanças ao nível arquitetural do sistema por forma a acomodar novos requisitos.
8. **Implementação das *User Stories* Refinadas**: escrita do código referente às novas funcionalidades requeridas.
9. **Testes às alterações arquiteturais**: teste e validação das mudanças introduzidas na estrutura do sistema.
10. **Testes de aceitação e compatibilidade**: planeamento e realização de testes, por forma a validar o trabalho desenvolvido.
11. **Escrita Relatório**: realização do relatório final de estágio, no qual vão estar documentadas todas as atividades desenvolvidas.



Name	Begin date	End date
Escolha das frameworks - Planeado	2/8/16	2/12/16
Escolha das frameworks - Real	2/8/16	2/9/16
Levantamento de Requisitos - Planeado	2/15/16	3/2/16
Requistos Funcionais - Elaboração das User Stories	2/15/16	2/24/16
Requisitos Não funcionais	2/22/16	3/2/16
Levantamento de Requisitos - Real	2/10/16	2/19/16
Requistos Funcionais - Elaboração das User Stories - Real	2/10/16	2/16/16
Requisitos Não funcionais - Real	2/15/16	2/19/16
Elaboração da Arquitetura - Planeado	3/3/16	3/23/16
Elaboração da Arquitetura - Real	2/22/16	3/4/16
Implementação das User Stories - Planeado	3/24/16	4/22/16
Implementação das User Stories - Real	3/7/16	4/25/16
Refinamento das User Stories - Planeado	4/25/16	5/3/16
Refinamento das User Stories - Real	4/26/16	4/29/16
Alteração da Arquitetura - Planeado	5/4/16	5/17/16
Alteração da Arquitetura - Real	5/2/16	5/13/16
Implementação das User Stories Refinadas - Planeado	5/18/16	6/6/16
Implementação das User Stories Refinadas - Real	5/13/16	6/7/16
Testes ás alterações arquiteturais - Planeado	6/8/16	6/10/16
Testes ás alterações arquiteturais - Real	6/8/16	6/10/16
Testes de aceitação - Planeado	5/18/16	6/6/16
Testes de aceitação - Real	5/13/16	6/7/16
Testes de compatibilidade - Planeado	6/10/16	6/15/16
Testes de compatibilidade - Real	6/10/16	6/20/16
Escrita do relatório - Planeado	2/8/16	7/1/16
Escrita do relatório - Real	2/8/16	8/26/16

Figura 3.2: Planeamento e tempo despendido nas tarefas do 2º Semestre.

3.2.4 Desvios ao planeamento do 2º semestre

Na Figura 3.2 é possível identificar a vermelho os desvios ocorridos ao planeamento, que aconteceram maioritariamente no processo de desenvolvimento e resultaram num atraso em algumas tarefas. Os atrasos de desenvolvimento deveram-se à utilização de duas *frameworks* distintas e totalmente desconhecidas, levando a um processo de aprendizagem complexo sobre o funcionamento destas. Outro dos problemas ocorreu na integração com um componente desenvolvido previamente, que teve de ser configurado e colocado em funcionamento.

Já os atrasos verificados na realização dos testes de compatibilidade são o resultado da execução dos clientes em diferentes plataformas, dos quais surgiu a necessidade de correção a alguns detalhes de implementação.

3.3 Análise de Riscos

A todos os processos de desenvolvimento de software estão associados eventos negativos que podem levar à falha do projeto, sendo, por isso, necessário avaliar a probabilidade de acontecer este tipo de eventos e fazer planos de forma a mitigar o seu aparecimento [111]. A este processo de identificação antecipada de riscos e planeamentos de mitigação dá-se o nome de análise de riscos [113]. Esta é feita de forma a reduzir a probabilidade de determinado projeto falhar e consiste na identificação, qualificação e mitigação dos fatores que podem trazer adversidades ao projeto afetando, assim, o seu sucesso.

Probabilidade de ocorrência	<30% **Baixa**	30%-50% **Média**	50%-75% **Alta**	>75% **Elevada**
Impacto	**Baixo**: projeto não se encontra comprometido	**Médio**: projeto não se encontra comprometido, mas é necessário controlar a evolução do risco	**Alto**: o projeto pode ser comprometido se não forem tomadas medidas adicionais	**Elevado**: o projeto pode estar seriamente comprometido e é necessário controlar estes riscos da melhor forma
Intervalo de tempo	**Fase inicial** do projeto, nas primeiras semanas de desenvolvimento	**Fase intermédia** do projeto, num estágio mais avançado do desenvolvimento	**Fase final** do projeto, posterior ao desenvolvimento	

Tabela 3.1: Parâmetros de avaliação de riscos.

Os riscos deste projeto vão ser avaliados segundo os parâmetros da Tabela 3.1, onde é apresentado o impacto e probabilidade de ocorrência de um risco, bem como o intervalo de tempo onde é previsto que este possa ocorrer. Vão ser também apresentadas consequências e planos de mitigação para os riscos associados ao projeto.

3.3.1 Riscos associados ao projeto

Risco	As *frameworks* escolhidas não permitem a implementação de todos os requisitos identificados
Probabilidade	Baixa
Impacto	Elevado
Intervalo	Fase inicial
Consequência	Incumprimento dos requisitos
Plano de mitigação	Análise prévia e detalhada das tecnologias a utilizar

Tabela 3.2: Risco número um.

Risco	Elevada curva de aprendizagem das *frameworks* escolhidas
Probabilidade	Baixa
Impacto	Baixo
Intervalo	Fase inicial
Consequência	Atraso na implementação dos requisitos
Plano de mitigação	Análise prévia e detalhada das tecnologias a utilizar

Tabela 3.3: Risco número dois.

Risco	Possíveis problemas de integração entre componentes
Probabilidade	Baixa
Impacto	Alto
Intervalo	Fase intermédia
Consequência	Impossibilidade da conclusão do projeto
Plano de mitigação	Estudo prévio da possibilidade de integração

Tabela 3.4: Risco número três.

Risco	Necessidade de alterar os requisitos do sistema
Probabilidade	Média
Impacto	Elevado
Intervalo	Fase intermédia
Consequência	Atraso no cumprimento de requisitos, realização de alterações a nível de implementação
Plano de mitigação	Reuniões de progresso periódicas, implementação simples o suficiente para acomodar facilmente alterações

Tabela 3.5: Risco número quatro.

Risco	Estimativas de implementação demasiado otimistas
Probabilidade	Média
Impacto	Médio
Intervalo	Fase intermédia
Consequência	Atraso no cumprimento de requisitos ou incumprimento dos mesmos
Plano de mitigação	Adequação dos tempos alocados para cada requisito

Tabela 3.6: Risco número cinco.

Risco	Não aprovação da aplicação multiplataforma por parte da *Ubiwhere*
Probabilidade	Baixa
Impacto	Alto
Intervalo	Fase final
Consequência	Refazer a implementação dos clientes
Plano de mitigação	Reuniões de progresso periódicas

Tabela 3.7: Risco número seis.

Risco	Necessidade de realizar alterações à arquitetura
Probabilidade	Média
Impacto	Elevado
Intervalo	Fase intermédia
Consequência	Atraso ou mesmo incumprimento de requisitos
Plano de mitigação	Adequar o tempo restante de acordo com a prioridade dos requisitos funcionais, arquitetura modular o suficiente para acomodar facilmente alterações

Tabela 3.8: Risco número sete.

Risco	Atualizações nas *frameworks* utilizadas que impliquem alterações na aplicação multiplataforma desenvolvida
Probabilidade	Média
Impacto	Alto
Intervalo	Fase final
Consequência	Alterações à aplicação multiplataforma de forma a acomodar as alterações realizadas na *framework*
Plano de mitigação	Estruturar o sistema por forma a permitir atualizações com o mínimo de esforço possivel

Tabela 3.9: Risco número oito.

Priorização de riscos

Os riscos do projetos identificados nas tabelas 3.2 à 3.9 foram priorizados tendo em conta a estratégia *Pareto Top N*, possibilitando a identificação dos riscos mais urgentes a resolver considerando a probabilidade de ocorrência, o impacto para o projeto e o intervalo de ocorrência previsto.

Probabilidade/Impacto	**Exposição do risco**
Alta/Baixo, Média/Baixo, Média/Médio, Baixa/Baixo, Baixa/Médio, Baixa/Alto	Baixa
Elevada/Baixo, Alta/Médio, Média/Alto, Baixa/Elevado	Média
Elevada/Médio, Elevada/Alto, Alta/Alto, Alta/Elevado, Média/Elevado	Alta
Elevada/Elevado	Elevada

Tabela 3.10: Exposição de riscos.

Na Tabela 3.10 é possível identificar quais os riscos mais significativos para o projeto, baseados em quatro níveis de análise. Ao combinar a exposição de determinado erro com o intervalo de tempo em que este pode ocorrer, é possível determinar a priorização dos riscos associados ao projeto.

	ID	**Probabilidade**	**Impacto**	**Exposição**	**Intervalo**
	4	Média	Elevado	Alta	Fase intermédia
25%	7	Média	Elevado	Alta	Fase intermédia
	1	Baixa	Elevado	Média	Fase inicial
	8	Média	Alto	Média	Fase final
	2	Baixa	Baixo	Baixa	Fase inicial
	3	Baixa	Alto	Baixa	Fase intermédia
	5	Média	Médio	Baixa	Fase intermédia
	6	Baixa	Alto	Baixa	Fase final

Tabela 3.11: Priorização de riscos segundo exposição e intervalo de tempo.

A priorização de riscos apresentada na Tabela 3.11 foi calculada segundo a exposição de cada risco e o intervalo de tempo onde se espera que este ocorra. Ainda na tabela anterior, pretende-se mitigar os riscos com probabilidade de ocorrência alta ou elevada e impacto alto ou elevado, resultando assim em 25% dos riscos identificados.

Dos riscos identificados e priorizados anteriormente ocorreram três: o risco quatro, o sete e o três. Neles são identificados riscos associados às alterações de implementação/arquitetura e integração de componentes.

Relativamente aos riscos ocorridos, houve a necessidade de alteração dos requisitos do sistema (risco 4), resultando em mudanças nas *User Stories* e nos atributos de qualidade (requisitos não funcionais). A introdução destas mudanças refletiu-se no acrescento e alteração de funcionalidades do sistema, e, por consequente, na alteração da arquitetura (risco 7). As mudanças efetuadas na arquitetura provocaram a adição e configuração de novos componentes, levando a problemas de integração (risco 3). Estes fatores desencadearam atrasos no processo de desenvolvimento e resultaram num esforço adicional para ultrapassar as dificuldades encontradas.

Capítulo 4

Especificação técnica

Este capítulo tem como objetivo apresentar o conjunto de pressupostos e detalhes técnicos utilizados para a construção do sistema de automático de *Digital Signage* multiplataforma, descrevendo requisitos e decisões arquiteturais tomadas.

4.1 Validação das plataformas de *deploy*

De forma a validar as plataformas iniciais e futuras de *deploy* de aplicações multiplataforma, é de seguida apresentado um estudo de mercado que identifica os principais sistemas operativos utilizados na atualidade.

No estudo da *Vision Mobile* [83], denominado *Developer Megatrends H1 2015* [15], é apresentada a segmentação de mercado em plataformas *desktop*, plataformas *mobile*, *smart TV*, *smart homes*, *werables* e *connected cars*. Desta segmentação de mercado surge a necessidade de responder a algumas perguntas fundamentais como:

- O produto deve ser desenvolvido para o sistema operativo X e Y em vez de outros?
- O produto deve ser desenvolvido só para os sistemas operativos mais populares?
- O produto deve ser o mesmo para o sistema operativo X e Y ou o produto deve ser diferente de sistema operativo para sistema operativo?
- Deve ser construída uma aplicação nativa ou uma aplicação web?

A resposta a este tipo de perguntas vai influenciar diretamente o processo de desenvolvimento do produto, nomeadamente os fatores de tempo de desenvolvimento, custo e *ROI* (*Return Of Investment*). As *frameworks* de desenvolvimento multiplataforma surgiram de forma a facilitar a resposta a este tipo de perguntas, ou seja, abstraindo o sistema final em que o produto irá executar e possibilitando a implementação de aplicações que estarão disponíveis para múltiplas plataformas.

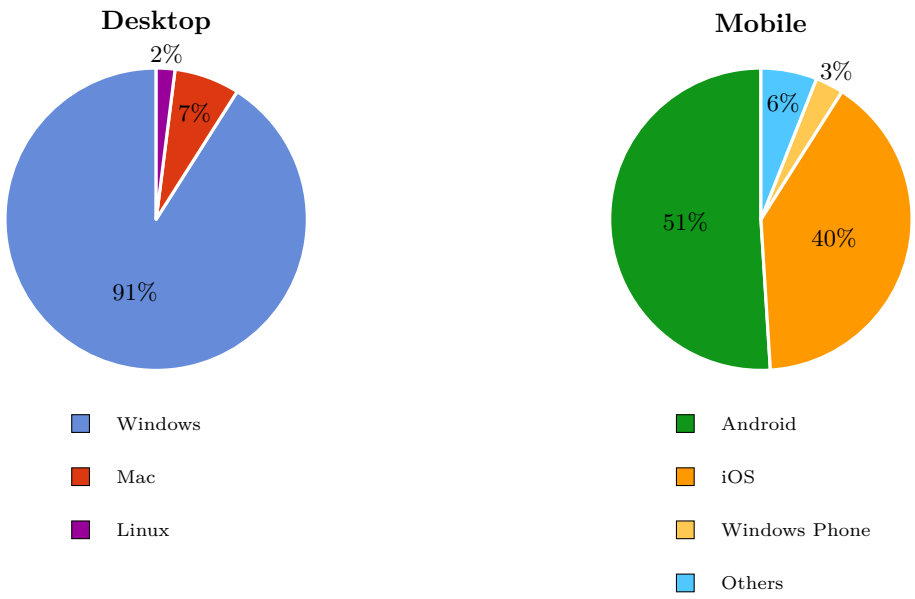


Figura 4.1: *Operating system market share*

Na figura Figura 4.1 são apresentadas as cotas de mercado por sistema operativo nos segmentos *desktop* e *mobile*, segundo os dados da *MarketShare*[47]. Analisando os dois gráficos é possível verificar que os sistemas operativos mais significativos são *Windows, Mac, Android* e *iOS*.

Apesar de neste estudo não serem apresentados resultados para os novos segmentos de mercado (*smart homes*, *werables*, *smart TVs* e *connected cars*), este possibilitou a validação das plataformas inicias de *deploy* do sistema. Ao descobrir quais as plataformas com maior quota de mercado, foi possível identificar os sistemas operativos críticos onde a aplicação multiplataforma deve estar disponível.

4.2 Requisitos do sistema

Nesta secção vão ser apresentados os requisitos aos se teve de dar resposta. Encontram-se divididos em: requisitos funcionais, referentes às funcionalidades do sistema; e requisitos não-funcionais, que dizem respeito ao uso do sistema propriamente dito.

4.2.1 Requisitos Funcionais

As funcionalidades deste projeto foram especificadas utilizando a narração da interação de um utilizador com o sistema, recorrendo a *User Stories* (US), seguindo o seguinte esquema:

- **Numeração das *User Stories*:** US.ID-Categoria.ID *User Story*.
- **Descrição:** Como <tipo de utilizador>quero <acção a realizar>de forma a <objectivo>.

- **Critérios de aceitação:** A funcionalidade está verificada se <lista de critérios>.

A utilização deste modelo de especificação deve-se à sua facilidade de construção e mapeamento em termos de testes de aceitação, bem como por ser próprio da metodologia ágil (SCRUM) utilizada durante este projeto. Relativamente às *US*, são referentes ao ator dispositivo, o qual representa as duas aplicações *desktop* e *mobile*.

O estagiário foi responsável pela elaboração das US com base no levantamento das necessidades da empresa. A cada requisito (US) foi atribuída uma prioridade na escala de 1 (prioridade mínima) a 5 (prioridade máxima). Esta priorização foi posteriormente validada pela empresa e foi definida como medida de sucesso a implementação de todos os requisitos, com prioridade entre 3 e 5. No decorrer do projeto, foi feito o refinamento das US, originando algumas alterações às *US* inicialmente definidas.

De seguida, é apresentado um resumo (numeração e descrição) das *US* que representam as funcionalidades de maior importância para o projeto.

- **US.1.4**: Como dispositivo quero receber os *settings* definidos no servidor para configurar o meu estado de funcionamento.
- **US.1.6**: Como dispositivo quero receber o agendamento de conteúdos para posterior exibição nos tempos definidos.
- **US.1.8**: Como dispositivo quero receber mudanças de agendamento para refletir estas mudanças em termos de exibição.
- **US.1.9**: Como dispositivo quero receber mudanças de conteúdos para refletir estas mudanças em termos de exibição.
- **US.1.10**: Como dispositivo quero receber mudanças de *settings* para refletir estas mudanças em termos de funcionamento.
- **US.1.13**: Como dispositivo quero exibir conteúdos recebidos para que estes sejam visualizados.

A lista completa e detalhada das *User Stories* deste projeto pode ser consultadas no Apêndice A, em concreto nas tabelas B.1 à B.5.

4.2.2 Requisitos não funcionais

Os atributos de qualidade ou requisitos não funcionais estão relacionados com o uso do sistema, representando critérios de análise para o funcionamento deste e ainda características mínimas que um *software* de qualidade deve cumprir.

Nesta Secção vão ser apresentados os requisitos não funcionais considerados fulcrais para o sistema automático de distribuição/exibição de conteúdos remotos. De notar que os requisitos apresentados são essenciais de garantir quando o sistema se encontrar num ambiente de produção. Contudo, para a prova de conceito

demonstrada neste relatório, a escalabilidade não vai ser alvo de medição. Com isto em mente, na Secção 5.2 é detalhado o conjunto de características e decisões tomadas durante a realização da prova de conceito para promover os seguintes requisitos não funcionais:

- **Atualização do agendamento dos clientes multiplataforma em *real-time***

  Este requisito é referente à capacidade do sistema propagar alterações nos agendamentos definidos e persistidos para os clientes multiplataforma do sistema. Posto isto, o sistema deve ser capaz de transmitir uma alteração no agendamento de um cliente no momento que esta acontece. Os clientes multiplataforma devem ser capazes de receber e reagir a esta alteração durante todo o seu período de funcionamento.

  Como se trata de um sistema para a apresentação de conteúdos multimédia em espaços públicos, onde estes podem ter um carácter publicitário, torna-se indispensável o carácter temporal da sua disponibilidade. Nesta caso, para que uma publicidade esteja disponível no intervalo de tempo certo e mal termine deixe imediatamente de o estar, deve ser utilizada comunicação em tempo real.

- **Escalabilidade**

  Este requisito é referente à capacidade do sistema executar normalmente quando se verifica um aumento de utilizadores ou clientes multiplataforma no sistema. Posto isto, o sistema deve ser capaz de garantir os recursos necessários para um normal funcionamento.

- **Resiliência a falhas de componentes**

  Este requisito é referente à capacidade de funcionamento do sistema perante falhas de componentes. Posto isto, a falha de um componente não deve afetar o normal funcionamento do sistema.

- **Segurança de utilização e comunicação**

  Este requisito é referente à capacidade do sistema garantir segurança, no que toca à sua utilização e em termos da comunicação entre cliente-servidor. Posto isto, o sistema só pode ser acedido por utilizadores autorizados e a transmissão de informações tem de ser feita de maneira segura.

- **Compatibilidade**

  Este requisito é referente à capacidade de dois sistemas trabalharem em conjunto, sem que tenham de ser alterados para isto. No que toca ao nosso sistema, compatibilidade é a capacidade de executar as aplicações responsáveis pela exibição em múltiplas plataformas.

Nota: Optou-se pelo detalhe dos requisitos anteriores numa Secção (5.2) posterior à apresentação da arquitetura do sistema (Seção 4.3) e detalhes de implementação (Seção 5.1) de forma a contribuir para uma melhor interpretação de como estes foram alcançados.

4.3 Arquitetura do sistema

Nesta secção vai ser apresentada a arquitetura proposta para o sistema automático de distribuição/exibição de ficheiros multimédia, para os sistemas operativos *Windows*, *Linux*, *Mac OS*, *iOS* e *Android*.

A arquitetura proposta foi criada tendo em conta um dos objetivos do estágio, que era a integração e modificação de uma plataforma existente para gestão e distribuição de conteúdos. A plataforma existente foi fornecida pela *Ubiwhere*, na forma de um servidor central que teve de ser instalado e configurado devidamente para perceber o conjunto de capacidades oferecidas. Após um período demorado e complexo de adaptação e aprendizagem do componente fornecido, foi possível determinar que este já contava com algumas das capacidades necessárias para a implementação do sistema a desenvolver, sendo elas:

- **Capacidade de gerir dispositivos**: o servidor permite a gestão de dispositivos ativos para exibição de conteúdos. Para o sistema a desenvolver, e sendo este direcionado para lojas de retalho, é importante controlar que dispositivos podem exibir conteúdos, na medida que, quando um é vendido, fica indisponível para a visualização de multimédia.
- **Capacidade de gerir/atribuir *playlists* de ficheiros multimédia**: o servidor permite a criação, edição e remoção de listas de conteúdos e intervalos de tempo de visualização e a atribuição destas listas a dispositivos ativos. Para o sistema a desenvolver, esta é uma das capacidades mais importantes, pois permite definir o conjunto de ficheiros/agendamentos a ser exibidos pelos clientes remotos.
- **Capacidade de gerir/atribuir *settings***: o servidor permite a criação, edição e remoção de conjuntos específicos de *settings*, que podem ser atribuídos aos dispositivos ativos. Para o sistema a desenvolver, esta capacidade é importante na medida que permite definir e atribuir parâmetros específicos para configurar o funcionamento dos clientes remotos.

Além destas características, foi possível determinar que o servidor expõe uma *REST API* (em detalhe na Seção 4.3.1) como método único de comunicação com os clientes remotos, o que implica o uso de *Client-pull* (descrito na Subseção 4.3.2), para obtenção de alterações aos ficheiros a exibir por determinado cliente (dispositivo). Para o sistema a desenvolver, este método de comunicação representa um problema em termos da propagação e reação a alterações de ficheiros/agendamentos em *real-time*, implicando por isso, a realização de mudanças ao componente recebido, de forma a garantir esta característica.

Posto isto, no seguimento desta secção, vão ser apresentadas duas arquiteturas: uma inicial, que consistiu na integração do componente recebido (*Django Back-end* (servidor)) com os clientes *desktop* e *mobile* a desenvolver, descrevendo estes componentes internamente; e uma final, onde são identificadas as alterações efetuadas à arquitetura inicial, de forma a garantir a característica de comunicação em *real-time* necessária pelo sistema a desenvolver.

4.3.1 Arquitetura inicial

Nesta secção é apresentada a arquitetura inicial do sistema, tendo em conta a integração do componente servidor (*Django back-end*) com os clientes multiplataforma responsáveis pela exibição de conteúdos, cliente *desktop* (*Windows*, *Linux*, *Mac OS*) e cliente *mobile* (*iOS* e *Android*). Detalha-se os constituintes internos de cada componente, o método de comunicação cliente-servidor (*client-pull*) e os modelos de dados utilizados para persistir e transmitir informação.

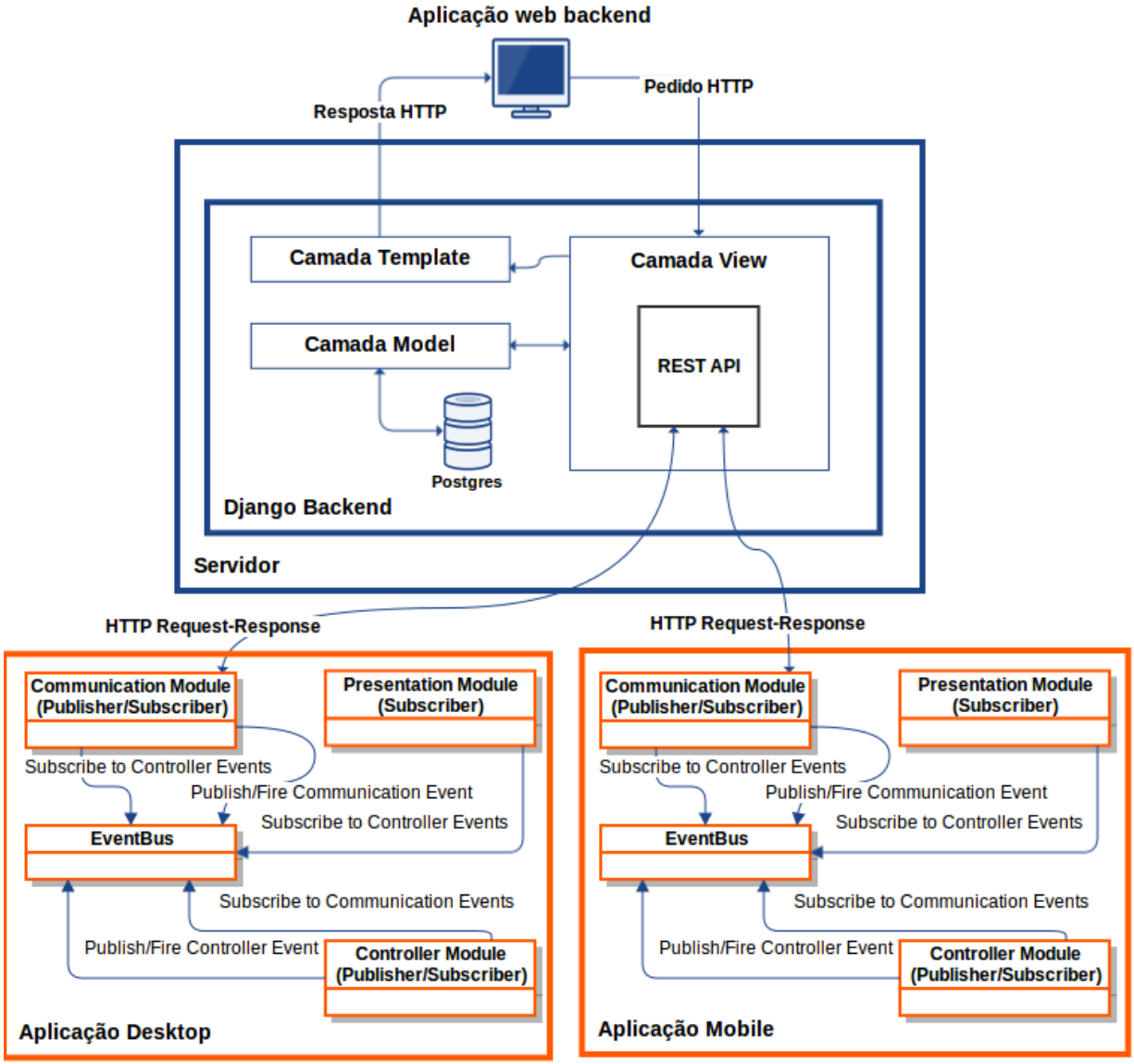


Figura 4.2: Arquitetura do sistema inicial

Na figura 4.2 encontra-se a arquitetura do sistema inicial. É possível observar os constituintes dos vários componentes e o método de comunicação usado entre eles. A laranja estão os componentes criados de raiz e a azul os componentes fornecidos pela empresa *Ubiwhere*. No seguimento desta secção, cada componente

apresentado na figura anterior vai ser analisado em detalhe.

Django back-end

Django é uma *framework* de alto nível para *Python* que permite o rápido desenvolvimento de aplicações *web*, facilitando as tarefas mais comuns relacionadas com o desenvolvimento *web*. Algumas das tarefas facilitadas por esta *framework* são:

- Interação entre a aplicação e a base de dados utilizando *object-relational mapping* (ORM), permitindo representar uma tabela da base de dados como um objeto em *Python*.
- Definição de vistas (página *html* a apresentar) acedendo a determinado URL, utilizando ficheiros de configuração simples e intuitivos.
- Visualização de dados em *HTML*, utilizando *templates* para apresentar dados de objetos mapeados.
- Autenticação de utilizadores, permitindo prover a aplicação *web* de controlo de utilizadores com o sistema de contas que vem incluído por defeito nesta *framework*.
- Administração do sistema, permitindo a criação automática de uma interface de administração baseada nos modelos de dados da aplicação.
- Segurança do sistema, provendo proteção contra: *SQL injection, Cross-site scripting, Cross Site Request Forgery*, entre outros.

Esta *framework* segue o conceito de *loosely coupling*[90], com a separação específica dos constituintes de uma aplicação baseada no padrão *Model View Controller* (MVC)[54]. Contudo, os criadores do *Django*, afirmam que esta segue um padrão *Model Template View (MTV)*[29]. Esta afirmação deve-se ao facto de em *Django* um *Template* representar a camada de apresentação (V), e de a *View* representar a camada que faz a "*bridge*"entre a camada *Template* e a camada *Model*, descrevendo que dados devem ser apresentada. Já a camada *Controller*, é considerada toda a *framework Django*[29]. Estas camadas estão ilustradas na figura 4.2, e a sua descrição é a seguinte:

- **Model**: Representa a camada de acesso à base de dados e é nesta que é feito o mapeamento *ORM* da base de dados *Postgres* utilizada.
- **Template:** Camada referente à apresentação de dados. No caso do sistema a desenvolver, corresponde à informação apresentada pela aplicação *web* de *back-end*.
- **View:** Camada responsável pela lógica de negócio. Esta camada é responsável por preparar toda a informação requerida pelo sistema. É nesta camada que é exposta a *REST API* (descrita na secção seguinte), que permite a comunicação entre clientes e servidor.

REST API

A *API REST* exposta pelo servidor representa o método principal de troca de informação entre servidor e cliente. Esta expõe um conjunto de *endpoints*, que ao serem chamados com um determinado verbo *HTTP* (*GET* ou *POST*) e com um conjunto de parâmetros específicos retornam o resultado de uma ação feita pelo servidor. Esta *API* torna possível que o cliente se encontre num estado sincronizado com o estado persistido no servidor.

Os *endpoints* definidos nesta *REST API* são os seguintes:

Verbo HTTP	**URI**	**Parâmetros**	**Descrição**
POST	/api/endpoints/register/	Token e Position	Permite o registo de novos clientes (dispositivos) no sistema, identificados pelo identificador único *Token*
GET	/api/endpoints/token/schedule/	Token e Active	Permite obter a programação actual (active=true), ou todas as programaçõs (active=false) de um dispositivo identificado pelo Token
POST	/api/endpoints/token/schedule/lock/	Lock e Token	Permite bloquear/desbloquear (lock=true or false) o *endpoint* anterior para só retornar a ação do servidor quando existem mudanças em determinado cliente identificado pelo Token
GET	/api/endpoints/token	Token	Permite retornar as configurações do cliente (dispositivo) identificado pelo Token.
POST	/api/endpoints/token/lock/	Lock e Token	Permite bloquear/desbloquear (lock=true or false) o *endpoint* anterior para que este só retorne a acção do servidor quando existem mudanças no cliente identificado pelo Token.

Tabela 4.1: REST API endpoints

O formato de dados usado por esta *API* é o formato *JSON*, onde um objeto é constituído por um conjunto de atributos e valores. Os dados disponíveis através desta *REST API* são o resultado do tratamento de dados provenientes da camada *Model* feito pela camada *View* do servidor. È através desta API que os clientes *desktop* e *mobile* obtêm o conjunto de ficheiros (*playlists*) a exibir, o seu agendamento e *settings* de configuração de funcionamento. Como tal, é importante referir as estruturas de *JSON* utilizadas para representar estes dados, podendo por isso, ser consultadas no Apêndice B. Ainda em termos das estruturas de dados utilizadas, no Apêndice C é possível encontrar o diagrama físico da base de dados *Postgre* utilizada

pelo servidor *Django*.

Comunicação cliente-servidor

Já com o modelo e estruturas de dados definidos é importante referir a sequência de pedidos efetuados entre os clientes *desktop* ou *mobile* e o servidor *Django back-end*. Esta sequência de pedidos representa os *HTTP Request-Response* da Figura 4.2 trocados entre o *Communication Module* e a *API REST* do servidor, de forma a tornar possível a exibição de conteúdos remotos.

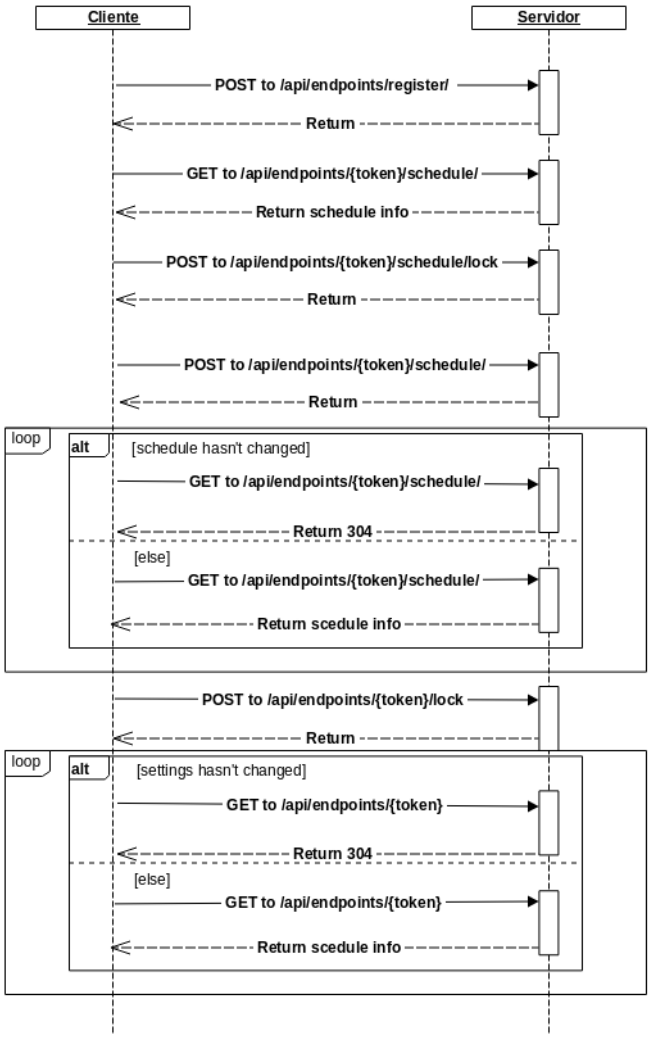


Figura 4.3: Diagrama de sequência da comunicação *client-pull*

O diagrama da figura 4.3 representa a sequência de pedidos trocados entre cliente e servidor. Neste é possível observar, que a comunicação é iniciada no cliente, e que o servidor "responde"com o resultado de determinada ação. De realçar, que no diagrama apresentado é possível identificar duas estruturas de repetição, com as suas respetivas condições. Estes dois *loops* representam a troca periódica de pedidos, para sincronização do

estado do cliente com o estado do persistido no servidor. São estes pedidos periódicos que permitem aos clientes reagir a alterações aos ficheiros multimédia a exibir, e que representam o problema à implementação de comunicação em tempo real (em detalhe na Subseção 4.3.2).

Componentes aplicação *Desktop* e aplicação *Mobile*

Os componentes cliente *desktop* e cliente *mobile* presentes Figura 4.2 representam as duas aplicações multiplataforma de visualização de conteúdos multimédia, construidas com recurso às *frameworks* escolhidas na Secção 2.4.2. As duas aplicações têm uma constituição/funcionamento semelhantes e diferem apenas na tecnologia usada para as criar. (*Apache Cordova* para a *mobile* e *Electron* para a *desktop*).

As duas aplicações são constituídas por quatro componentes: três módulos, *Communication Module*, *Controller Module* e *Presentation Module* e um agregador de eventos, *EventBus*. Os módulos seguem os padrões de *Javascript Module Pattern*[44] e uma variação do *Observer Pattern*[43] denominado por *Publish/Subscribe*. O primeiro padrão, *Module pattern* foi utilizado para emular em *Javascript* o conceito de classes com métodos privados e públicos. Já o segundo padrão, *Publish/Subscribe* foi utilizado em conjunto com o *EventBus* de forma a implementar uma comunicação entre módulos baseada num sistema de eventos que permite evitar dependências de componentes. A descrição destes quatro constituintes é a seguinte:

- ***Communication Module***: Este módulo é responsável por toda comunicação com o servidor. Para isto, realiza pedidos *HTTP* à *REST API* e publica *Communication events* para o *EventBus* com o resultado dos pedidos efetuados. Este pedidos são efetuadas através da subscrição de *Controller Events*, ou seja, o *Controller Module* é que dita que pedidos devem ser efetuados.

- ***Controller Module***: Este módulo é responsável tanto por publicar *Controller events* para o *EventBus*, como por executar as funções correspondentes aos *Communications events* subscritos. E pode ser visto como a ponte entre o *Communication Module* e o *Presentation Module*, pois consoante os dados recebidos vai enviar o *Controller Event* necessário para alterar o comportamento do *Persentation Module*. É neste modulo que são controladas todas as trocas de ficheiros a exibir, bem como a persistência/leitura de ficheiros locais.

- ***Presentation Module***: Este módulo é o responsável por apresentar no ecrã, os conteúdos impostos pelo *Controller Module*, ou seja, é apenas responsável por executar a função correspondente ao *Controller Event* publicado para o *EventBus*.

- ***EventBus***: Este constituinte serve apenas o papel de agregador de eventos, disponibilizando a todos os módulos a possibilidade de *subcribe*/*unsubscribe* e *publish* de eventos. O funcionamento do *EventBus*, consiste no registo de funções a executar (subcribe) para determinado evento, e quando este acontece (*publish*) são simplesmente chamados todos os *listeners* registados para o evento.

Com esta descrição em mente, a baixo é apresentado o diagrama de classes que descreve os quatro componentes já analisados internamente.

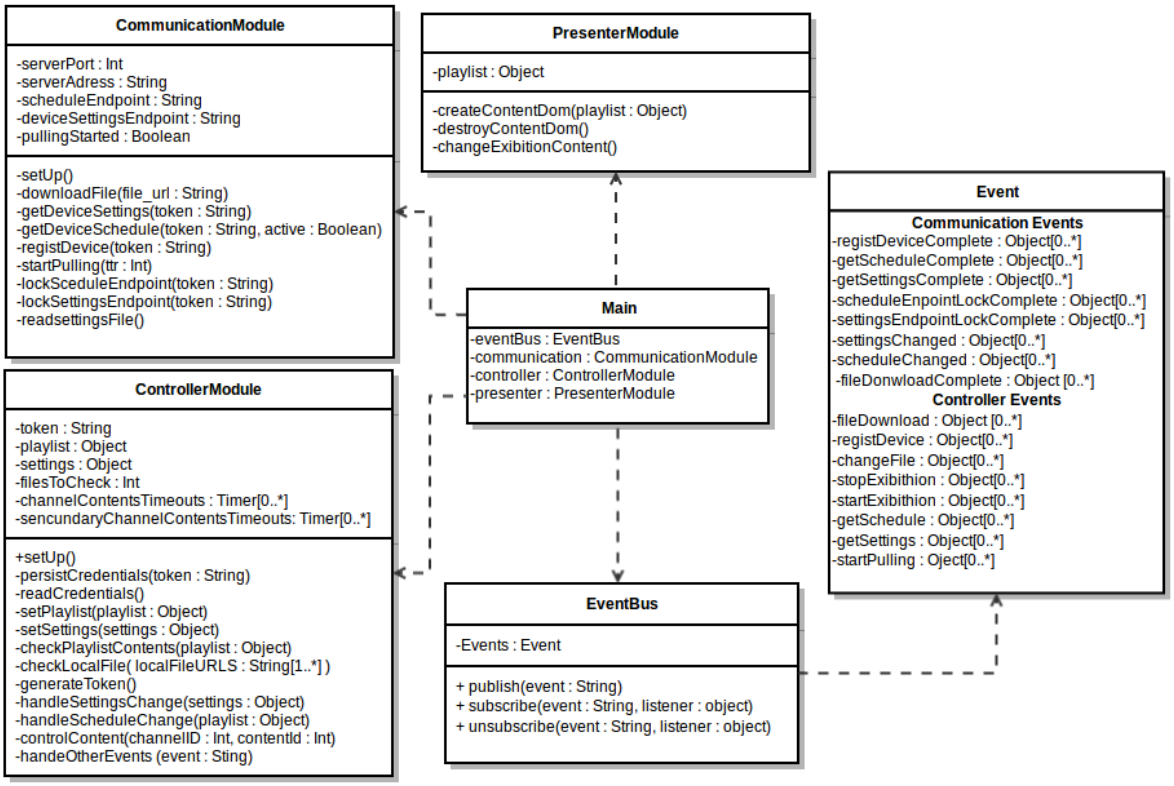


Figura 4.4: Diagrama de classes.

No diagrama de classes presente na figura Figura 4.4 é possível observar os métodos e atributos de cada módulo criado, assim como do agregador de eventos (*EventBus*). Ao analisar esta figura é possível reparar que a maioria dos métodos são privados, com a exceção das funções *publish*, *subscribe* e *unsubcribe* do *EventBus* e a função *setUp* do *Controller Module*. Em relação à publicidade de todos os métodos do *EventBus*, isto deve-se ao facto de todos os módulos puderem disponibilizar *listeners* para qualquer evento (subscribe), assim como publicar dados para estes (publish). Já no que diz respeito ao método publico *setUp* do *Controller Module*, isto è necessário para controlar especificamente o inicio da aplicação, pois é este módulo que despoleta a da exibição de conteúdos. Por último, todos os restantes métodos são privados pois são despoletados internamente ao módulos após a receção do evento que lhe corresponde no *EventBus*. Para terminar, e ainda tendo em conta a figura anterior, pode-se constatar que todos os módulos são independentes, o que facilita alterações e até mesmo a inserção de novos módulos. Num exemplo concreto, é possível criar um novo módulo de comunicação mantendo o funcionamento da aplicação, bastando que o módulo criado implemente os mesmos eventos que o *Communication Module*.

Como já referido, a comunicação entre módulos é efetuada com o recurso a um sistema de eventos baseado

no padrão *Publish/Subscribe*. Esta comunicação pode ser consultada no Apêndice D na forma de diagramas de sequência descritivos dos vários eventos subscritos e despoletados pelos módulos constituintes das duas aplicações multiplataforma.

4.3.2 Arquitetura Final

Alteração do método de comunicação *Client-pull* para *Server-push*

Na arquitetura do sistema inicial descrita na secção 4.3.1, o método de comunicação utilizado era o *Client-pull*. Neste tipo de comunicação o cliente comunica com o servidor em intervalos de tempo pré-definidos (*Time to Refresh* (TTR)), de forma a atualizar o seu estado. Esta comunicação ocorre mesmo que o estado do cliente não tenha mudado, o que pode originar uma troca de mensagens desnecessárias. De forma a garantir que o cliente se encontre num estado atualizado com o persistido no servidor é necessário que a frequência de *pulling* (TTR baixo) seja alta, originando assim um tráfego de rede elevado [95].

No sistema a desenvolver, os conteúdos multimédia apresentados pelos clientes multiplataforma podem ser do tipo publicitário, onde, são por norma apresentados os detalhes de uma promoção. Normalmente, uma promoção está disponível em determinado período de tempo, e quando esta termina os clientes devem deixar de a apresentar quase de imediato. Assim com o método de comunicação *Client-pull*, para que os clientes reajam a esta alteração o mais rápido possível, é necessário um TTR muito baixo implicando o aumento de pedidos feitos ao servidor. Intuitivamente, quantos mais clientes ativos no sistema, maior vai ser o número de pedidos feitos ao servidor. Implicando assim uma maior carga computacional em termos de tráfego de rede e processamento do servidor.

No método de comunicação *Server-push*, o cliente não necessita de efetuar pedidos periódicos para atualizar o seu estado, pois é possível comunicar alterações para os clientes sem que estes as tenham de pedir especificamente. Este tipo de comunicação pode ser utilizado para implementação de sistemas em *real-time* (ex : *instant messaging*), possibilitando assim que os clientes do nosso sistema reajam em tempo real a alterações de estado no servidor. Facto este que é importante, por exemplo, no caso da apresentação de um conteúdo publicitário já referido. Assim ao utilizar este método de comunicação é possível:

- Reduzir a carga computacional do servidor, em termos de processamento de pedidos desnecessários.
- Reduzir o tráfego de pedidos na rede.
- Reduzir as dependências de componentes do sistema.
- Diminuir complexidade de implementação dos clientes multiplataforma.
- Desenvolvimento de clientes com capacidades de reação em "*real-time*".

- Possibilidade de uso de topologias de integração como *Message Bus* ou *Message Broker*.

Com isto em mente, foram estudadas alternativas para a implementação do método de comunicação *Server-push* na arquitetura inicial do sistema de forma a possibilitar a propagação de alterações em *real-time*. O estudo de alternativas foi efetuado de forma a que possíveis alterações ao sistema já desenvolvido mantivessem a possibilidade de utilização de *Client-pull*, bem como, resultassem em poucas mudanças no que toca à implementação do servidor *Django* e das aplicações *mobile* e *desktop*.

Normalmente, os interveniente de uma comunicação *Server-push* têm de exprimir preferências da informação a receber antecipadamente. Isto, é chamado de modelo *Publish/Subscribe*. Neste modelo, o cliente subscreve a canais de informação disponibilizados por um servidor; e quando existe um novo conteúdo destinado a um desses canais o servidor faz o *push* dessa informação para os clientes que o subscreveram [66]. Estes canais são baseados em conexões permanentes entre cliente-servidor, e uma maneira segura de os implementar é com recurso a *Websockets*.

Posto isto, foi analisada a possibilidade de implementar *Websockets* no componente *Django back-end* como forma de comunicação em *real-time* para com as aplicações *mobile* e *desktop*. Ao estudar esta alternativa deparou-se que a implementação de *Websockets* no servidor é um processo complexo devido a *framework Django* utilizada para o seu desenvolvimento.

O *Django* foi construido sobre o conceito *request-response* e não permite a criação de conexões permanentes (*long living connections*) [33]. Tornando-o inapropriado para a criação de sistemas em que os clientes têm de reagir a eventos iniciados no servidor (*Server-push*) devido à impossibilidade de envio de informações sem que esta seja pedida [36]. Contudo, *Django* pode ser utilizado em paralelo com outro serviço responsável pelos *Websockets* para a implementação do modelo *Publish/Subscribe* necessário à comunicação *Server-push* [86]. Bastando que estes serviços possibilitem a receção de informações através de pedidos HTTP e que disponibilizem meios para as transmitir para os *Websockets* apropriados.

Atendendo à explicação anterior, foram consideradas as seguintes tecnologias a integrar na arquitetura inicial do sistema por forma a implementar a comunicação *Server-push*:

- ***Pusher***[19] é uma *event-based API* hospedada na web, que permite a comunicação em *real-time* através de *Websockets*. Esta *API* disponibiliza uma *REST interface*, que permite a publicação de informações através de pedidos *HTTP*, bem como um biblioteca para *Javascipt* que facilita a sua utilização nesta linguagem. Estes dois aspetos fazem de *Pusher* uma tecnologias de fácil integração no nosso sistema, pois as duas aplicações (*dektop* e *mobile*) foram implementadas em *Javascript* e é possível a comunicação com este serviço através de pedidos *HTTP* provenientes do servidor *Django*.

- ***Crossbar.io***[14], utiliza a implementação de *Websockets* com *Web Application Messaging Protocal* (*WAMP*) fornecida pelo projeto *open-source autobahn.ws* para disponibilizar comunicação em *real-time* baseada em *Publish/Subscribe*. Esta tecnologia utiliza o conceito de *cross-protocol*, para permitir que múltiplos dispositivos comuniquem entre si, funcionando como um *router* de mensagens. Como a tecnologia anterior, *Crossbar.io* também pode ser utilizado para receber pedidos *HTTP* do servidor *Django* e propagar dados em *real-time* para as aplicações *desktop* ou *mobile*.

- ***Meshblu***[46] é um projeto *open-source* para comunicação instantânea entre dispositivos. Este, disponibiliza *HTTP*, *Websockets*, *MQTT* ou *COAP* como possíveis protocolos para transmissão de mensagens. Viabilizando a implementação de Publish/Subscribe entre dispositivos que utilizam protocolos de comunicação diferentes. Como as tecnologias anteriores, este também pode ser utilizado para receber pedidos *HTTP* do servidor *Django* reencaminhando-os para conexões permanentes criadas pelas aplicações responsáveis por exibir conteúdos multimédia.

A possibilidade de utilização destas tecnologias foi posteriormente apresentada à empresa *Ubiwhere*, por forma, a perceber qual delas seria a mais vantajosa para a integração no sistema de distribuição/exibição de conteúdos automático. Desta apresentação, surgiu o facto de que o *Meshblu* já era utilizado na empresa com fins semelhantes, prover comunicações em *real-time*. Levando assim à escolha desta tecnologia, devido ao suporte adicional oferecido pela *Ubiwhere* perante eventuais problemas de integração.

Mediante a explicação anterior vai ser apresentada a arquitetura final para prover comunicação em *real-time* de alterações ao estado dos clientes multiplataforma (*mobile* e *desktop*).

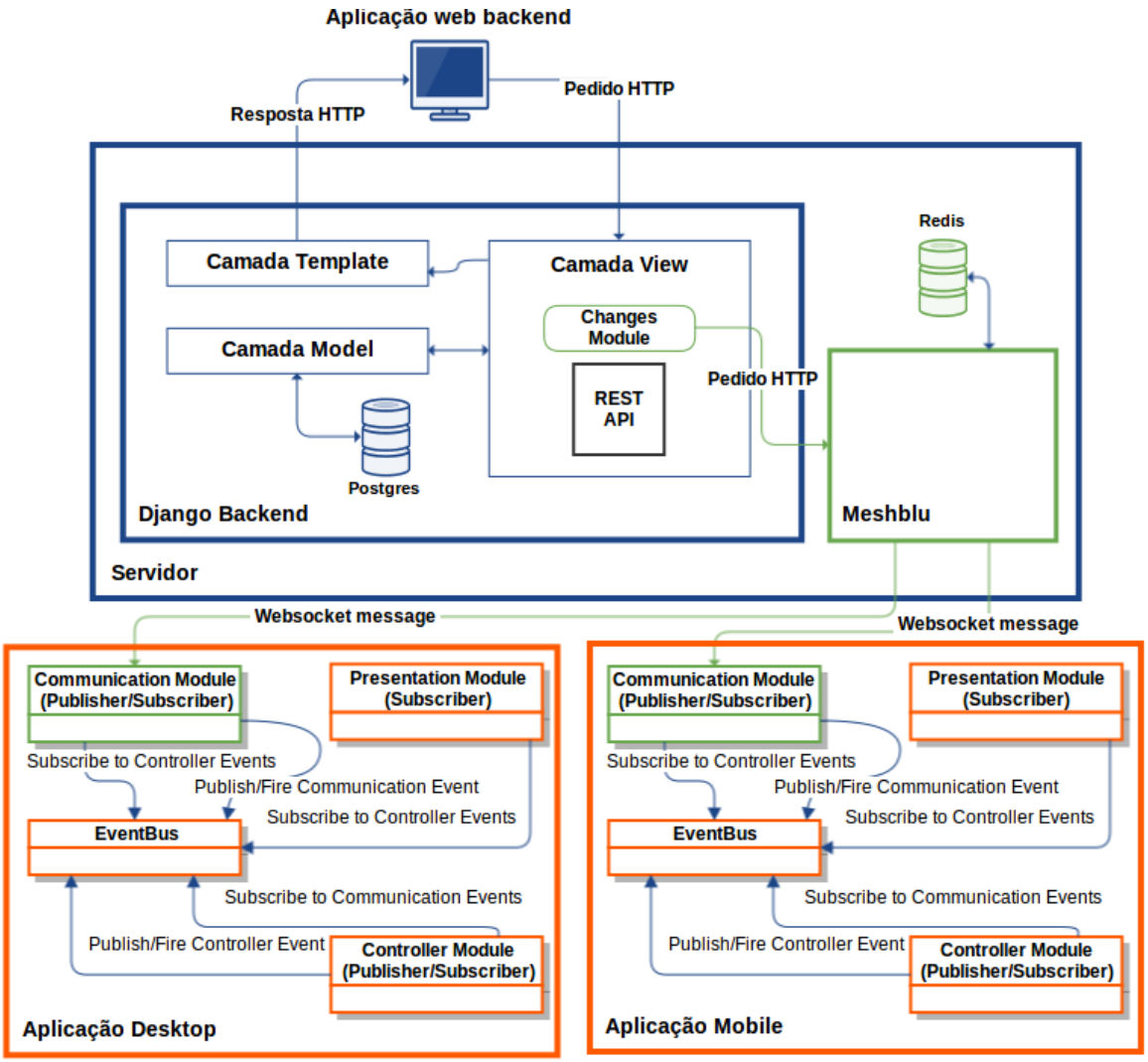


Figura 4.5: Arquitetura final.

Na Figura 4.5 a verde é possível identificar a adição de novos componentes, *Meshblu*, *Redis*, *Changes Module*, alteração ao método de comunicação entre cliente e servidor, comunicação *HTTP* para comunicação baseada em *Websockets*, assim como mudanças no *Communication Module*. Estas alterações são referentes à arquitetura inicial apresentada na Figura 4.2 da secção 4.3.1.

Changes Module

Como já enunciado, com *Server-push* a comunicação é iniciada pelo servidor e eventuais mudanças de estado de um cliente são comunicadas sem que este as tenha de pedir especificamente. Para tornar isto possível, o servidor têm de ser capaz de reconhecer automaticamente mudanças de estado e de as propagar para os clientes. Para isto, foi adicionado o módulo *Changes module* á camada *View* do referido servidor. A funcionalidade principal deste módulo é a criação/envio de notificações automáticas para o componente *Meshblu*,

de forma a que este as possa propagar para os clientes ativos no sistema na forma de *Websocket messages*.

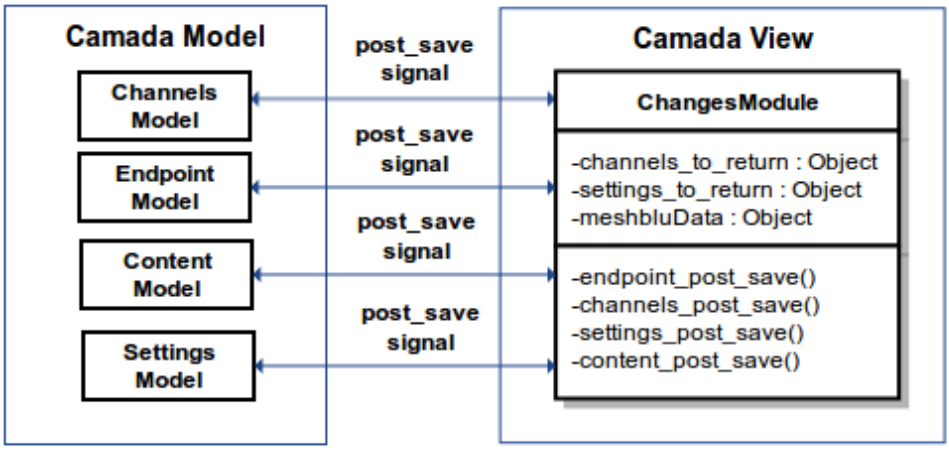


Figura 4.6: *Changes Module* em detalhe

Na Figura 4.6 é apresentado em detalhe os constituintes do *Changes Module* adicionado ao componente *Django Backend* para possibilitar a comunicação em tempo real com os clientes do sistema. È possível observar que este módulo é utilizado por quatro modelos de *Django*, em concreto, *Channels*, *Endpoint*, *Content* e *Settings Model*. Estes modelos despoletam funções do módulo criado através da utilização do sistema *Signaling*[71] oferecido pela *framework Django*. Para isto, quando um destes quatro tipos de modelos é persistido na base de dados é enviado um sinal de *post_save* para uma função destino do *Changes Module*, a qual, por sua vez, vai processar o conjunto de alterações realizado e clientes (aplicação *mobile* ou *desktop*) afetados. Após este processo, é criada/enviada uma notificação para o componente *Meshblu* com todos os dados necessário para que os clientes afetados possam reagir à alteração efetuada no servidor.

Em termos dos dados retornados por este módulo, optou-se pelo mesmo formato utilizado pela *REST API*, com a adição de dois campos *type* e *devices*. Estes campos descrevem o tipo de notificação e dispositivos afetados. No Apêndice B é possível consultar um exemplo de uma notificação retornada pelo *Changes Module* e na Seção 5.1 encontra-se uma explicação mais detalhada dos tipos de notificações utilizados, bem como, as restantes funcionalidades associadas a este módulo.

Meshblu e Redis

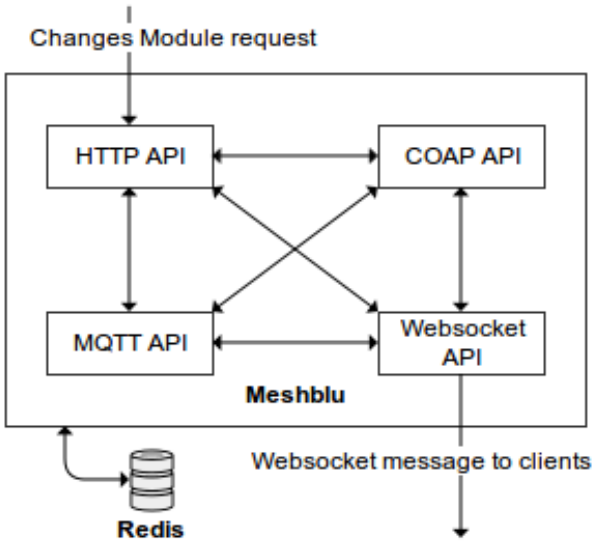


Figura 4.7: *Meshblu.*

Na Figura 4.7 são apresentados os protocolos constituintes do *Meshblu*, *HTTP*, *COAP*, *MQTT* e *Websockets*. É possivel observar que todos estes protocolos são disponibilizados através de *APIs* e que estas comunicam entre si. Dos protocolos representados os menos conhecidos são o *COAP* e *MQTT* pois estão associados à nova área de *Internet of Things* (*IoT*) e servem o propósito de comunicação com dispositivos de baixa capacidade de computação (ex: sensores, micro-controladores), não sendo por isso relevantes para este projeto. Já as *APIs* para os protocolos *HTTP* e *Websockets* foram as utilizadas para prover o sistema com comunicações em tempo real.

As *APIs* disponibilizadas por este componente só podem ser acedidas utilizando um conjunto de credenciais únicas (*Universally Unique Identifier* (*UUID*) e *Token*) que permitem identificar inequivocamente os intervenientes de comunicações a efetuar. Para isto, o *Meshblu* disponibiliza um sistema de registo onde são gerados e devolvidos o conjunto de *UUID* e *Token* associados a determinado dispositivo. E, estas credenciais são posteriormente persistidas numa *Key-Value datastore* denominada por *Redis* para utilizações futuras sem um novo registo.

Este conjunto de identificadores únicos (UUID e Token) permitiu a implementação do padrão de comunicação *Publish/Subscibe* necessário à implementação de comunicação em tempo real no sistema a desenvolver. Neste, tanto o *Django back-end* como todos clientes *Mobile/Desktop* são registados no *Meshblu*. E posteriormente, os *UUIDs* atribuídos aos clientes são propagados para o servidor *Django* de forma a tornar possível a geração de notificações por parte do *Changes Module*.

Como já referido, o *Changes Module* cria/envia notificações para os clientes do sistema. Para isto, são computadas as alterações a efetuar, bem como, o conjunto de clientes afetados na forma de uma lista de *UUIDs*. É nesta lista de *UIIDs* que são identificados todos os *Subscribers* que devem receber a notificação permitindo que o *Meshblu* faça o *Store* ou *Forward* de uma mensagem recebida. Em concreto, quando uma notificação tem como destino um dispositivo ativo no sistema o *Meshblu* faz o *Forward* da mensagem para o *Websocket* apropriado e caso contrário (dispositivo offline) é feito o *Store* da mensagem no *Redis* para posterior envio.

Como forma adicional de segurança o *Meshblu* permite a definição de permissões de envio e receção de mensagens entre dispositivos. Possibilitando a definição de listas de UIID's para os quais um dispositivo pode enviar ou receber mensagens. Um exemplo da utilização deste sistema de permissões pode ser encontrado no Apêndice E onde é apresentado o envio de uma notificação para dois clientes com configurações de segurança diferentes.

Alterações ao *Communication Module* das aplicações *Desktop* e *Mobile*

Como já referido as duas aplicações multiplataforma *desktop* e *mobile* foram criadas tendo em conta dois padrões de *Javascript*, *Module Pattern* e uma variação do *Observer Pattern* conhecida como *Publish/Subscribe*. Foi nesta fase da implementação do método de comunicação *Server-Push* mantendo a possibilidade de utilização de *Client-Pull* que a utilização de padrões de desenvolvimento se revelou uma grande vantagem na integração dos clientes multiplataforma com o componente *Meshblu*. Pois em vez de ser necessário a alteração de todos os constituintes das aplicações multiplataforma por forma a acomodar uma comunicação baseada em *Websockets* apenas o *Communication Module* (módulo responsável pela comunicação com serviços externos) teria de ser reformulado.

Apesar de apenas com a reformulação do *Communication Module* ser possível garantir o funcionamento dos dois métodos de comunicação *Server-Push* e *Client-Pull* optou-se por implementar um novo módulo denominado por *Meshblu Communication Module*. Este módulo é dedicado à comunicação em tempo real e visa promover a independência dos constituintes das duas aplicações multiplataforma. Para o seu desenvolvimento apenas foi necessário garantir que um conjunto de eventos já definidos (Figura D.1 do Apêndice D) fossem despoletados para manter o funcionamento da solução já implementada.

Já no que toca à possibilidade de utilização da comunicação *Client-Pull*, o novo módulo *Meshblu Communication Module* possibilita a utilização do *Communication Module* quando existe algum erro de conexão com o componente *Meshblu*. Permitindo o *Pulling* ao servidor *Django* de forma a manter o funcionamento da solução desenvolvida perante a impossibilidade do uso de comunicação em tempo real.

A baixo encontra-se o diagrama de classes que diz respeito às duas aplicações multiplataforma.

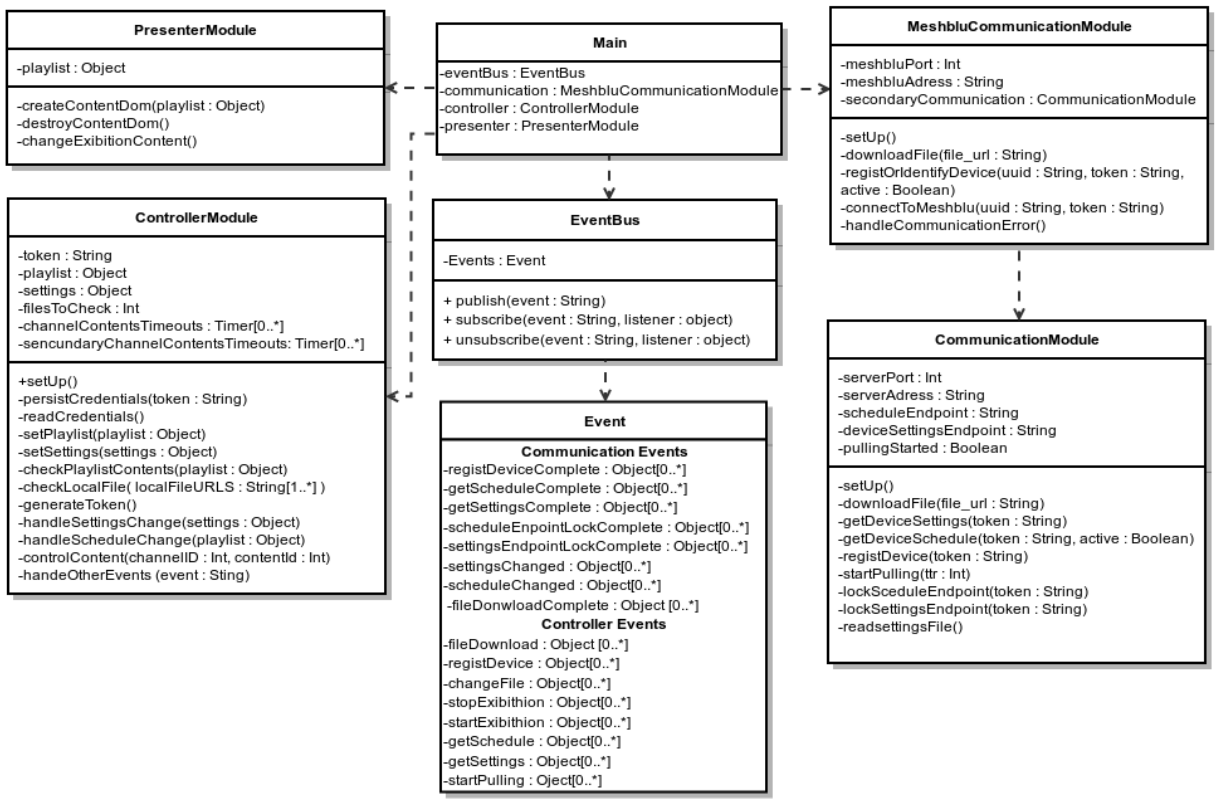


Figura 4.8: Diagrama de classes final.

Na figura Figura 4.8 encontra-se o diagrama de classes final dos dois clientes multiplataforma *desktop* e *mobile*. É possível observar a adição de um novo módulo, *Meshblu Communication Module* e a alteração do já existente *Communication Module* para funcionar como dependente do novo módulo de comunicação em caso de erros. Estas mudanças são referentes ao diagrama de classes presente na Figura 4.4 da Secção 4.3.1.

Ainda na figura anterior um aspeto importante a referir é a complexidade reduzida do novo módulo de comunicação desenvolvido (*Meshblu Communication Module*). Como os dados não têm de ser especificamente pedidos ao servidor e não é necessário fazer pedidos periódicos para atualização de estado, este módulo só tem de ser capaz de receber *Websocket Messages* do componente *Meshblu* e despoletar *Communication Events* já definidos para desencadear a exibição de conteúdos.

> **Nota:** No Apêndice D podem ser consultados os diagramas de comunicação entre módulos referentes ao diagrama de classes da Figura 4.4 da Secção 4.3.1, contudo a comunicação entre módulos referente ao diagrama de classes final apresentado na Figura 4.8 da presente Secção não é descrita pois o novo módulo introduzido (*Meshblu Communication Module*) segue o mesmo modelo de comunicação da Figura D.1 do já referido Anexo, apenas não despoletando eventos desnecessários.

Capítulo 5

Detalhes de implementação e testes realizados

Neste capítulo vão ser detalhados os aspetos a realçar da implementação efetuada para atingir a solução final já descrita. Também se vão dar a conhecer os principais testes efetuados durante toda a duração deste projeto de estágio e, ainda, como são promovidos os requisitos não funcionais identificados na Subseção 4.2.2.

5.1 Implementação

Durante o desenvolvimento do sistema já descrito, foi necessário ter em conta alguns aspetos para conseguir atingir alguns dos requisitos determinados. Como tal, vão ser apresentados de seguida como foram alcançados alguns desses objetivos, detalhando alguns aspetos de implementação.

5.1.1 *Changes Module*

O *Changes Module* foi o componente adicionado ao servidor *Django* para possibilitar comunicações *Server-push*. Este módulo é responsável por detetar alterações as configurações dos dispositivos de exibição de conteúdos e de as propagar na forma de notificações para o *Meshblu*. E, as características a destacar são as seguintes:

- **Geração de listas de dispositivos afetados:** Uma alteração, pode resultar numa mudança de estado para múltiplos clientes. Assim, antes da geração de uma notificação, é criada uma lista com todos os dispositivos afetados;

- **Geração de notificações:** Ao ser detetada uma alteração, é criada uma notificação única que a descreve. Esta notificação vai conter uma lista de todos os clientes afetados, assim como, todos os

campos necessários para que estes reajam à alteração do seu estado realizada no servidor. As notificações criadas seguem um formato *JSON* com a seguinte estrutura:

- ***type***: descreve o tipo de alteração efetuada pela aplicação de *back-end*, podendo tomar os seguintes valores:
  - ***device_playlist_changed***: representa a alteração as listas de reprodução atribuídas a um dispositivo. Este tipo de notificação é apenas referente a um dispositivo.
  - ***device_settings_changed***: representa uma alteração aos *settings* atribuídos a um dispositivo. Este tipo de notificação é apenas referente a um dispositivo.
  - ***playlist_changed***: representa uma alteração a uma *playlist* do sistema. Este tipo de notificação pode ser referente a vários dispositivos, pois a mesma *playlist* pode estar atribuída a múltiplos dispositivos.
  - ***settings_changed***: representa uma alteração a um conjunto de *settings* do sistema. Este tipo de notificação pode ser referente a vários dispositivos, pois o mesmo conjunto de *settings* pode estar atribuído a múltiplos dispositivos.
  - ***content_changed***: representa uma alteração a um conteúdo presente no sistema. Este tipo de notificação pode ser referente a vários dispositivos, pois um conteúdo pode estar presente em várias *playlists*, e uma *playlist* em vários dispositivos.
- ***devices***: lista com a identificação, dos dispositivos afetados pela alteração ocorrida, podendo também ser vista como os destinatários da notificação criada. Nos casos em que a alteração afete apenas um dispositivo, a lista é populada com apenas um elemento.
- ***data***: os dados propriamente ditos resultantes da alteração efetuada. Estes dados diferem para cada tipo de notificação, e são genéricos o suficiente para descreverem uma alteração no estado de múltiplos dispositivos.

  Nota: Um exemplo de uma notificação com esta estrutura *JSON* pode ser consultada no Apêndice B.

- **Envio automático de notificações:** Após a geração de uma notificação, esta é enviada automaticamente para o componente *Meshblu*, para que seja propagada até aos dispositivos afetados. Estes envios foram feitos recorrendo a um biblioteca denominada por *requests*[69], a qual permite fazer pedidos HTTP(S) em *Python*.

- **Prevenção de erros:** As funcionalidades deste módulo são despoletadas, após a persistência com sucesso, de uma alteração no *Postgres*. Permitindo, que os clientes só sejam notificados de alterações que se processaram sem qualquer tipo de erro. A deteção da correta persistência de de dados foi feita utilizando as capacidades de *Model Signaling* oferecidas pelo *Django*.

Um aspeto importante da implementação deste módulo foi a necessidade da compreensão aprofundada dos constituintes do servidor fornecido pela *Ubiwhere*, bem como da *framework* de desenvolvimento *Django*. De maneira a preservar o funcionamento de uma solução em fase de produção da empresa. Com esta tarefa em mente, e de forma a manter a implementação original o mais inalterada possível optou-se pela construção do descrito *Changes Module* para prover o sistema de comunicações em tempo real. A implementação deste módulo permitiu manter uma parte essencial do projeto quase inalterada, os modelos *ORM* de *Django*. Este facto deveu-se a utilização das funcionalidades de *Model Signaling*, que permitiram executar funções especificas quando um modelo de *Django* è corretamente persistido na base de dados *Postgre*.

5.1.2 As duas aplicações multiplataforma

Para a construção do sistema de visualização automático de conteúdos multimédia nos sistemas operativos *Windows*, *Mac OS*, *Linux*, *Android* e *iOS*, foram implementadas duas aplicações multiplataforma. Uma para o segmento *desktop* utilizando a *framework Electron*, e outra para o segmento mobile com recurso a *framework Cordova*. Estas duas *frameworks*, apesar de completamente distintas têm como base o desenvolvimento de aplicações utilizando a linguagem de programação *Javascript*.

O facto anterior levou à possibilidade de utilização de um *single shared code base* para a implementação das duas aplicações multiplataforma. Para isto, durante a implementação das duas aplicações multiplataforma foi dada preferência ao uso de funções e objetos de *Javascript* nativos, ou seja, pertencentes à especificação *standard ECMAScript*[22]. Contudo, para a realização de algumas tarefas (acesso ao sistema de ficheiros do dispositivo) foi necessário empregar funcionalidades especificas das *frameworks Electron* ou *Cordova*.

Com isto em mente, e, apesar de não existir um *single shared code base* propriamente dito, a reutilização de código entre a versão *mobile* e *desktop* das aplicação multiplataforma é de grande escala. E foi possível, através da escolha de tecnologias de desenvolvimento apropriadas e decisões de implementação atempadamente tomadas (utilização de *Javasript* Nativo).

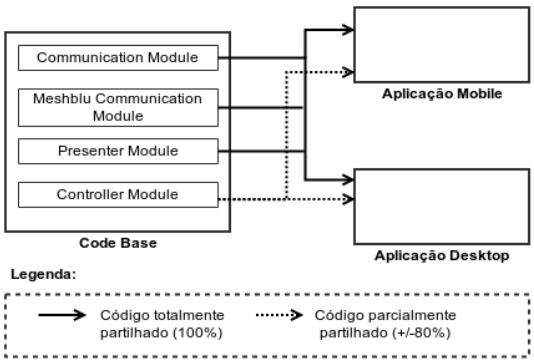


Figura 5.1: Reutilização de código.

Na Figura 5.1 é apresentado a reutilização de código conseguida entre versões. É possível observar que a maioria da implementação efetuada é reutilizavél entre versões. Em concreto, o código dos módulos *Communication Module*, *Meshblu Communication Module* e *Presenter Module* é totalmente(100%) partilhado entre versões e a maioria (80%) do código que constitui o *Controller Module* também é partilhado.

Atendendo à explicação anterior de seguida vão ser descritos todos os módulos presentes na Figura 5.1, detalhando aspetos de implementação e referindo como foi conseguida a sua reutilização nas duas versões da aplicação multiplataforma (*mobile* e *desktop*).

Communication Module

Communication Module, módulo responsável pela comunicação *Client-Pull* da arquitetura inicial do sistema apresentada na Secção 4.3.1. Neste módulo são feitos todos os pedidos à *REST API* exposta pelo servidor *Django*. Pedidos estes que consistem em *HTTP Requests* com o verbo *POST* ou *GET*, para determinado *endpoint*. Para a implementação destes pedidos optou-se pela utilização da *API XMLHttpRequest*[92] nativa ao *Javascript*. O que permitiu que a total reutilização de código deste módulo entre a versão *mobile* e *desktop* da aplicação multiplataforma.

Com a existência de um único *Communication Module* partilhado pelas versões *mobile* e *desktop* da aplicação em mente, algumas características a destacar são:

- **Validação dos dados recebidos**: os dados obtidos da *REST API*, são inspecionados por forma a rejeitar estrutura e conteúdos não válidos.
- **Validação da conclusão de pedidos HTTP**: o estado de realização de todos os pedidos efetuados é observado de forma a perceber/prevenir ocorrência de erros.
- **Prevenção de erros de comunicação**: quando um pedido HTTP falha por completo, este é repetido até o máximo de cinco vezes. Não afetando o funcionamento da solução final até o número de tentativas ser esgotado. Quando isto acontece, é propagado e tratado um evento de erro que terminará a exibição de conteúdos.

Meshblu Communication Module

Com a mesma finalidade do módulo anterior, obtenção de dados do servidor *Django*, o *Meshblu Communication Module* em conjunto com o *Changes Module* e *Meshblu* permitem comunicações em tempo real baseadas em *Server-Push* para com as aplicações multiplataforma *desktop* e *mobile*. Para isto, este módulo cria conexões permanentes com o componente *Meshblu* e fica posteriormente à espera de mensagens (notificações) geradas pelo *Changes Module*. Para a implementação das conexões permanentes optou-se pelo o uso da *Websocket API*[79] nativa de *Javascript* e descrita na especificação da *World Wide Consortium*[91].A utilização

desta API possibilitou que este módulo fosse totalmente partilhado entre as versões *mobile* e *desktop* da aplicação multiplataforma.

Este módulo constitui uma parte fundamental do projeto desenvolvido pois é um dos componentes essenciais à comunicação em tempo real no sistema final e as características a destacar são:

- **Validação dos dados recebidos**: os dados obtidos numa *Websocket Message*, são inspecionados por forma a rejeitar estrutura e conteúdos não válidos.
- **Interpretação das alterações recebidas**: quando são recebidos dados válidos do *Websocket*, o tipo da notificação recebido é interpretado de forma a enviar o *Communication Event* necessário para a alteração das configurações da aplicação.
- **Prevenção de erros de comunicação**: quando é perdida a comunicação com o *Websocket* ou existe algum erro de comunicação com o componente *Meshblu*, este módulo permite a utilização de *Client-Pull* (disponibilizada pelo Communication Module) para manter a aplicação num estado normal de funcionamento. E quando isto acontece, o *Meshblu Communication Module* inicia um *Ping* periódico com o componente *Meshblu* por forma a retomar o funcionamento da aplicação com comunicações *Server-push* em tempo real.

Presenter Module

Este módulo é responsável por apresentar no ecrã associado ao dispositivo os conteúdos multimédia a ele atribuídos por um gestor do sistema. Como a apresentação de multimédia, consiste na maioria das vezes em exibir listas de conteúdos seguindo uma ordem especifica e em diferentes regiões do ecrã, no *Presenter Module* é utilizado *jQuery* para a criação/manipulação de todos os objetos necessários á visualização de videos/imagens impostos pelo servidor.

Ao iniciar o período de funcionamento da aplicação, este módulo recebe o conjunto de ficheiros a exibir e cria toda a *Document Object Model* (*DOM*) necessária para a visualização dos vários tipos de multimédia. Após esta fase, fica á espera que o *Controller Module* gere o evento necessário para mostrar no ecrã o primeiro conteúdo multimédia. E quando este evento é recebido o *Presenter Module* modifica a visibilidade da DOM já criada para exibir o primeiro conteúdo, ficando á escuta de posteriores *Controller Events*. Nesta fase, quando um evento é recebido, pode ser alterado o ficheiro a exibir ou criada nova *DOM* para refletir eventuais alterações.

Em termos da compatibilidade deste módulo perante as aplicações mobile e *desktop* e atendendo ao uso de jQuery para manipulação de *DOM*, foi necessário o uso de duas versões desta biblioteca de *Javascript*. Em concreto, *jQuery Mobile* 1.4.5 na aplicação *mobile* e *jQuery* 2.2.2 na aplicação *desktop*. Apesar de terem

fins distintos, a utilização destas duas bibliotecas permitiu o uso do mesmo conjunto de funções nas duas aplicações multiplataforma tornando o *Presenter Module* totalmente reutilizável.

Controller Module

O ultimo módulo representado na Figura 5.1, denominado por *Controller Module*, é fundamental para o controlo do correto funcionamento da aplicação multiplataforma. É neste módulo que são geridos todos os ficheiros de configuração necessários, bem como, as trocas de dados entre módulos e ficheiros a exibir. Funcionando como uma ponte entre o módulo responsável pela comunicação, *Communication Module* ou *Meshblu Communication Module* e o módulo responsável pela apresentação de conteúdos, *Presenter Module*.

Em concreto, este módulo é responsável por ler/escrever ficheiros contendo conjuntos de credenciais e informações relacionadas com a localização do servidor *Django* e *Meshblu*. Bem como da persistência de ficheiros multimédia já obtidos do servidor, para utilizações futuras sem a necessidade de novo *download*.

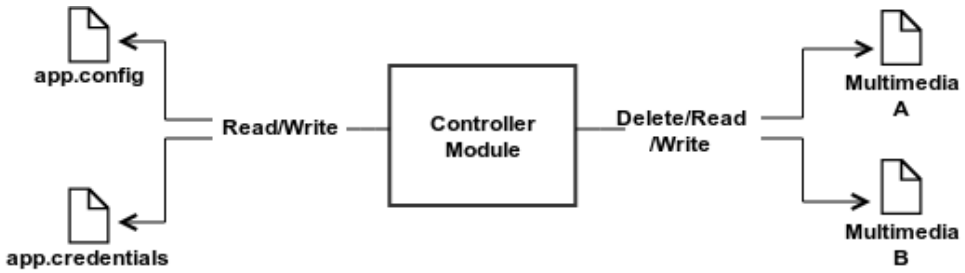


Figura 5.2: Ficheiros utilizados pela aplicação multiplataforma.

Na Figura 5.2 estão representados os ficheiros persistidos localmente utilizados pelo *Controller Module*. É possível observar que os ficheiros de configuração *app.config* e *app.credentials* podem apenas ser lidos ou escritos (*Read/ Write*) ao contrário dos ficheiros multimédia que também podem ser apagados (*Delete*).

Em termos da possibilidade de remoção de ficheiros multimédia, esta deve-se a questões relacionadas com falta de espaço de armazenamento. As quais são tratadas, eliminando os ficheiros não utilizados à mais de um dia, de forma a acomodar localmente os ficheiros mais pertinentes para a exibição de conteúdos por parte do *Presenter Module*. Já os ficheiros de configuração *app.credentials* e *app.config* não podem ser removidos, pois contêm todo o conjunto de informações indispensáveis ao funcionamento da aplicação.

> **Nota:** No Apêndice F, pode ser consultado em detalhe o conjunto de dados guardados nos ficheiros de configuração *app.config* e *app.credentials*.

A necessidade de utilização de ficheiros para manter dados entre execuções consequentes da aplicação multi-plataforma, resultaram na impossibilidade de reutilizar totalmente o *Controller Module* entre a versão *mobile* e *desktop*. Pois em *Javascript* nativo é impossível o livre acesso ao *File System* do sistema operativo subjacente.

Para resolver a impossibilidade de acesso ao sistema de ficheiros local, optou-se pela utilização de funcionalidades especificas das *frameworks* Electron e Cordova. Em concreto para a versão *desktop* da aplicação desenvolvida com recurso a *Electron* foi utilizado o pacote fs [31], e para a versão *mobile* um *plugin* de *Cordova* denominado *Cordova-plugin-file* [30].

- **Pacote fs**: API dedicada ao processamento de ficheiros, disponibilizando *wrappers* em *Javascript* para funções *POSIX standard*.
- ***Cordova-plugin-file***: este *plugin* disponibiliza o acesso ao sistema de ficheiros de um dispositivo móvel, abstraindo o sistema operativo que este está a executar. Tratando das diferenças entre os sistemas de ficheiros de iOS e Android.

A utilização destes dois recursos implicou que as funções responsáveis pelo manuseamento de ficheiros diferissem da versão *desktop* para a versão *mobile*. Tornado apenas 80% do código do *Controller Module* reutilizável.

Para além do manuseamento de ficheiros, este módulo é responsável por outras tarefas, das quais se destacam o seguinte conjunto:

- **Manter em memória a *playlist* em reprodução**: é neste módulo que está em memória o conjunto de ficheiros e referentes agendamentos atualmente atribuídos ao dispositivo.
- **Alteração do ficheiro a exibir**: é este módulo que controla o ficheiro exibido pelo *Presenter Module*, alterando o mesmo de forma a cumprir o agendamento imposto pelo servidor.
- **Reação a alterações**: é este módulo que permite alterar o conjunto de ficheiros a exibir, ou outras configurações, perante a receção de uma notificação no módulo de comunicação.
- **Gestão de download**: é este módulo que mediante um conjunto de verificações obtêm do módulo de comunicação eventuais ficheiros multimédia ainda não presentes localmente no dispositivo.

5.2 Requisitos não funcionais em detalhe

Os requisitos não funcionais são por norma mantidos por decisões a nível da arquitetura do sistema e implementação propriamente dita. Posto isto, nesta secção vai ser apresentado o conjunto de características e funcionalidades do sistema que permitiram alcançar os requisitos identificados na Subseção 4.2.2.

- **Atualização do agendamento dos clientes multiplataforma em *real-time***

  Para promover este requisito o sistema faz uso da comunicação *Server-Push* com recurso a funcionalidades automáticas de deteção e propagação de alterações a agendamentos. Em concreto, este requisito é alcançado através da seguinte caracteristica:

  - **Utilização de *Push-Notifications***: o sistema faz uso de *Websockets* disponibilizados pelo componente *Meshblu*, que permitem ao *Changes Module* o envio de uma *Websocket Message*(notificação) mal é detetada um alteração de agendamento destinada a um cliente do sistema.

- **Escalabilidade**

  As características do sistema que permitem garantir escalabilidade num eventual ambiente de produção originam essencialmente da arquitetura do sistema. Compreendendo decisões arquiteturais provenientes da inserção/integração do componente *Meshblu* com o servidor *Django* fornecido pela *Ubiwhere*, bem como as as suas características. Posto isto, os as aspetos que permitem garantir escalabilidade são:

  - **Utilização de um Message Broker (Meshblu)**: a introdução do componente *Meshblu* na arquitetura do sistema permite fazer o uso de uma topologia de integração desacopolando o servidor *Django* (*Sender*) e os clientes multiplataforma (*Receivers*). Aumentando assim o nível de integrabilidade, modificabilidade, segurança e testabilidade do sistema [49].
  - **Utilização de componentes escaláveis**: por si só, os componentes *Meshblu* e *Django* já promovem escalabilidade, sendo usados pela *Ubiwhere* em diferentes projetos em fase de produção. Além disto, estes dois componentes contêm as seguintes características:
    - ***Django*** promove a fácil adição de *hardware*, servidores de base de dados, servidores web, entre outros, o que permite escalar os recursos computação disponíveis.
    - ***Meshblu*** pode ser utilizado na *cloud*, em conjunto com serviços de *load balancing* e *auto scaling*, aumentando ou diminuindo os recursos disponíveis, de acordo com a carga computacional necessária. Possibilitando assim, servir um número variável de clientes multiplataforma.

- **Resiliência a falhas de componentes**

  O sistema garante resiliência perante falhas de componentes, na medida que disponibiliza formas de lidar com eventuais indisponibilidades de alguns componentes. Estas formas surgem essencialmente do tratamento de erros implementado e são as seguintes:

- **Utilização de dois métodos de comunicação**: quando a comunicação *Server-Push* é impossibilitada devido a uma falha do componente *Meshblu*, é possível a utilização de *Client-Pull* (comunicação directa com o Django) para manter a comunicação do sistema, e por consequente, o seu funcionamento;
- **Replicação parcial de dados**: Existe um conjunto parcial de dados dos clientes multiplatorma replicado na base de dados utilizada pelo *Meshblu* (*Redis*), permitindo o normal funcionamento do sistema, na impossibilidade de acesso a base de dados *Postgres*;

- **Segurança de utilização e comunicação**

  As características do sistema que permitem uma utilização e comunicação segura do sistema provêm essencialmente da implementação efetuada em termos de autenticação de utilizadores, uso de protocolos encriptados de comunicação e do uso das potencialidades de segurança oferecidas pelo componente *Meshblu*. Posto isto, o sistema promove este requisito não funcional, na medida que:

  - **Disponibiliza um sistema de *login***: as funcionalidades do sistema só podem ser acedidas inserindo um conjunto de credenciais válido, permitindo a utilização do sistema apenas por utilizadores autorizados;
  - **Disponibiliza de um sistema de permissões**: um utilizador do sistema só pode aceder a funcionalidades para as quais contém permissões, promovendo assim segurança de utilização.
  - **Permite troca segura de mensagens**: no que toca a comunicações efetuadas no sistemas, estas só podem ser efetuadas utilizando um conjunto válido de credenciais (uuid e token). E os intervenientes têm de estar identificados na lista de dispositivos autorizados (em cada dispositivo), para que exista comunicação.
  - **Utiliza comunicação encriptada**: as comunicações do sistema são efetuadas usando os protocolos seguros de comunicação *HTTPS* e *Websockets*.

- **Compatibilidade**

  Compatibilidade, uma das restrições inicias deste projeto de estágio, foi conseguida através do desenvolvimento de duas aplicações multiplataforma, uma para o segmento *mobile* e outra para o segmento *desktop*. Permitindo que os clientes do sistema responsáveis pela apresentação de conteúdos estejam presentes nos sistemas *Windows*, *Linux*, *Mac OS X*, *Android* e *iOS*.

5.3 Testes realizados

Esta secção tem como objetivo apresentar os principais testes realizados durante o desenvolvimento de todo o projeto.

5.3.1 Testes às alterações arquiteturais

Como forma da validar as alterações introduzidas na arquitetura inicial do sistema, a comunicação cliente-servidor foi monitorizada, utilizando a ferramenta *WireShark*. Assim, o número de pacotes trocados foi avaliado tendo em conta o método de comunicação *client-pull* (arquitetura inicial) *vs server-push* (arquitetura final). Como estes dois métodos de comunicação apresentam características diferentes, os seguintes parâmetros foram tidos em conta para os testes:

- **Tempo total de teste**: de forma a restringir o intervalo de tempo de monitorização.
- **Número total de alterações a inserir:** o número máximo de alterações a inserir durante o tempo total do teste.
- ***Time to Refresh*(client-pull)**: como já referido *client-pull* necessita a definição de um intervalo de tempo para realizar o *pulling* periódico ao servidor.

Tendo em conta estes parâmetros, foram efetuados os seguintes testes:

Pârametros	Teste 1	Teste 2	Teste 3	Teste 4
Tempo total de teste (min)	10	10	10	10
Número total de alterações a inserir	2	2	2	2
Time to Refresh (ms)	1000	10000	30000	60000

Tabela 5.1: Parâmetros de teste

Na Tabela 5.1, é possível observar que foram executados quatro testes ás arquiteturas do sistema, onde foram variados os tempos de *pulling* (TTR). O parâmetro *TTR* é apenas referente á arquitetura inicial do sistema e a variação que pode observada, tem com intuito descobrir o TTR aproximado, que pode ser usado na arquitetura inicial para que o número de pedidos efetuados se assemelhe aos efetuados na arquitetura final. Tendo em conta que, um *TTR* mais baixo implica um aumento de pedidos efetuados e um *TTR* mais alto implica uma diminuição de pedidos efetuados.

A inserção de alterações foi feita de forma aleatória, durante a execução de cada teste. Consistindo na mudança da *playlist* em exibição por um cliente multiplataforma *desktop*, já registado/autenticado perante o sistema e em atual exibição de conteúdos multimédia. Para simular a alteração de uma *playlist*, foi utilizada a ferramenta *Webdriver* a qual permite "imitar"a utilização da aplicação *web* de *back-end* por parte de um

utilizador. Todos os testes foram executados num ambiente local de desenvolvimento, onde se encontravam a correr respetivamente, a arquitetura inicial do sistema ou a arquitetura final do sistema. Para que as medições efetuadas pelo *Wireshark*, fossem o mais viável possíveis, a ligação a *Internet* foi cortada, desativando as interfaces responsáveis por esta.

No Apêndice G podem ser consultados os resultados das medições efetuadas tendo em conta a descrição anterior para os testes efetuados. Resultando no gráfico comparativo da troca de pacotes apresentado de seguida.

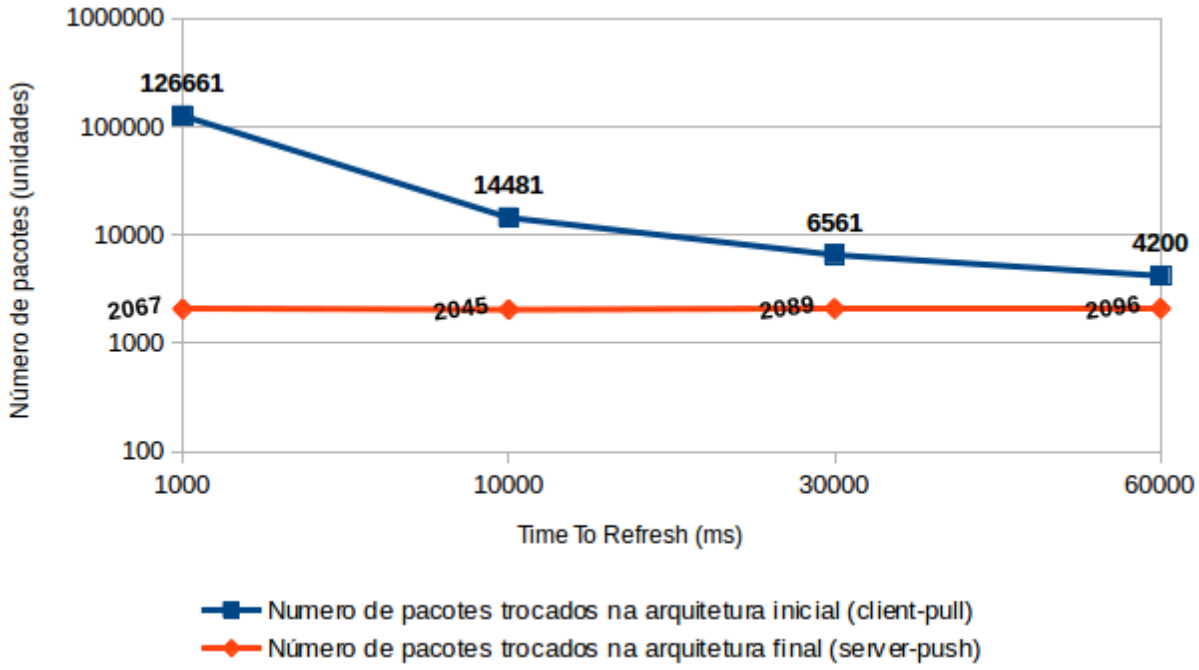


Figura 5.3: Gráfico comparativo do número de pacotes trocados.

No gráfico da Figura 5.3 é possível observar o número de pacotes trocados tendo em conta as duas arquiteturas de sistema já apresentadas. Verificando-se que, no caso da arquitetura inicial o número de pacotes trocados vária com o *TTR*, e na arquitetura final a variação do número de pacotes trocados, em cada teste, é mínima. Ainda com o gráfico anterior, é possível verificar que com *server-push* (arquitetura final), o número de pacotes trocados é muito inferior aos de *client-pull* (arquitetura inicial) com valores de *TTR* até sessenta mil milissegundos (um minuto).

Concluindo, para que a arquitetura inicial apresente um número de pacotes trocados próximo da arquitetura final, implica o uso de um TTR alto, o que por consequente, se reflete na capacidade de reação a alterações do cliente multiplataforma. Num exemplo concreto, com pedidos de *pulling* de um em um minuto (TTR = 60000ms), em que é realizada uma alteração a *playlist* em exibição do dispositivo. Se esta for processada,

logo após à resposta a um pedido de *pulling*, o dispositivo só conseguirá refletir a alteração da *playlist*, no próximo pedido de *pulling*, ou seja, um minuto depois. Já na arquitetura final, este problema não acontece, pois no momento que é detetada uma alteração, esta é propagada para o cliente, sem que esta a tenha de pedir especificamente, não necessitando, por isso, da troca periódica de pedidos, entre cliente-servidor.

Na arquitetura inicial, para conseguir capacidades de reação em *real-time* semelhantes às da arquitetura final, é necessário o uso de um *TTR* muito baixo, o que se reflete, num aumento da troca de pacotes, e consequentemente, no aumento da carga computacional do servidor e da rede. Assim, as alterações arquiteturais, efetuadas, resultaram na diminuição da carga de rede e trabalho computacional do servidor, para conseguir a capacidade de reação em *real-time*, por parte dos clientes muliplataforma.

5.3.2 Testes de Compatibilidade

Como já enunciado, um dos objetivos principais deste projeto, era a criação de clientes multiplataforma para os sistemas operativos *Windows*, *Mac OS X*, *Linux*, *Android* e *iOS*, sendo por isso importante testar a solução desenvolvida no que toca ao suporte destes sistemas operativos. Para isto, foram criados executáveis para as referidas plataformas, e o seu funcionamento foi testado/observado em várias versões do mesmo sistema operativo.

Em termos da geração de executáveis, para as plataformas *desktop* foi utilizado o pacote *electron-packager* [25] enquanto para as plataformas *mobile* foi utilizado o *Cordova CLI* [8]. Estas duas ferramentas disponibilizam uma interface na linha de comandos que permite a geração de executáveis para o conjunto de sistemas operativos suportados pelas *frameworks Electron* e *Cordova* respetivamente. Disponibilizando ainda, um conjunto de funcionalidades extra que permitem, entre outras, a escolha da arquitetura (x86 ou x64) ou ativação de métodos de compressão de código.

Plataformas *mobile*

Sistema Operativo	**Resultado do Teste**
Android 6.0	Passou
Android 5.1	Passou
Android 4.4	Passou
iOS 9	Passou
iOS 8	Passou
iOS 7	Passou

Tabela 5.2: Resultados dos testes para plataformas *mobile*.

Para testar a aplicação multiplataforma desenvolvida nos sistemas operativos mobile apresentados na Tabela 5.2 foram utilizados emuladores com as versões pretendidas para cada plataforma. Além dos emuladores, também foram utilizados dois dispositivos físicos com *iOS 7* e *Android 5.1*, de forma a testar a aplicação num ambiente de funcionamento mais "real". Para cada sistema a aplicação foi executada e o seu funcionamento observado, corrigindo e recomeçando o teste após a deteção de comportamentos inesperados.

Para todas as versões (6.0, 5.1 e 4.4) do sistema *Android* não foram verificados quaisquer erros de execução, contudo em todas as versões de *iOS* testadas, e apesar da aplicação apresentar um funcionamento correto, os conteúdos multimédia não eram exibidos no ecrã do dispositivo. Após identificar este desvio de comportamento, os objetos *HTML* responsáveis pela renderização de determinado ficheiro foram alterados e todos os testes para plataformas *mobile* recomeçados. E, no fim destes testes, foi possível verificar o mesmo comportamento e correto funcionamento da aplicação para todas as versões alvo das plataformas *mobile*.

Plataformas *desktop*

Sistema Operativo	**Resultado do Teste**
Windows 10	Passou
Windows 8.1	Passou
Windows 7	Passou
Linux Ubuntu 16.04	Passou
Linux Ubuntu 14.04	Passou
Linux Ubuntu 12.04	Passou
Mac OS X 10.11	Passou
Mac OS X 10.10	Passou
Mac OS X 10.9	Passou

Tabela 5.3: Resultados dos testes para plataformas *desktop*.

Como forma de simular os diferentes sistemas operativos presentes na Tabela 5.3, foram utilizadas as capacidades de virtualização oferecidas pela *VirtualBox*, com a exceção dos sistemas operativos *Windows 10* e *Linux Ubuntu 14.04* (ambiente de desenvolvimento) que se encontravam instalados localmente. E, da mesma forma que os testes de compatibilidade nas plataformas *mobile*, para cada sistema operativo a aplicação foi executada, e o seu funcionamento observado. Nos casos em que houveram desvios ao comportamento esperado, os erros detetados foram corrigidos e o teste recomeçado.

Em todos os sistemas operativos da tabela anterior, a execução da aplicação com o método de compressão ASAR[26] ativo (configuração *default* de *builds* construidos com *electron-packager*) apresentou um comportamento erradico. Onde por vezes era possível iniciar a aplicação e ela fazia "freeze", e por outras, não era sequer possível dar inicio ao seu funcionamento. De forma a resolver este comportamento, a compressão

ASAR foi desligada e todos os testes recomeçados.

Para além do contra tempo anterior, no sistema operativo Windows a execução da aplicação originava erros de acessos a ficheiros, o que impossibilitava o seu correto funcionamento. Após identificar que os problemas decorriam do uso de caminhos relativos para recursos foram feitas as correções necessárias e o comportamento esperado foi atingido.

Após as duas alterações anteriores, todas as versões alvo das plataformas *desktop* ficaram a funcionar corretamente.

5.3.3 Testes de aceitação

Como forma a determinar se os requisitos funcionais foram corretamente implementados durante todo o processo de desenvolvimento foram efetuados testes de aceitação. Para isto, sempre que se terminava a implementação de um requisito era verificado se o seu funcionamento correspondia ao definido nas *User Stories* presentes no Apêndice A. E caso se verificassem desvios ao funcionamento esperado procedia-se à realização das correções necessárias para que a funcionalidade fosse de encontro ao especificado.

Um aspeto importante de referir sobre estes testes, é a validação feita pela a empresa *Ubiwhere* de todas as funcionalidades desenvolvidas. Esta validação incidiu principalmente em duas fases da implementação do sistema. Na primeira fase, integração das aplicações multiplataforma *mobile* e *desktop* com o servidor *Django* foram validadas as funcionalidades mais importantes do projeto (prioridade 5 nas *US*), ou seja, o conjunto essencial de requisitos a implementar. Já na segunda fase, integração de comunicações em *real-time* com a solução já desenvolvida, todas as funcionalidades implementadas foram validadas, incidindo especialmente nas funcionalidades referentes à comunicação *Server-push* e provenientes da inserção do componente *Meshblu* na arquitetura do sistema.

Os resultados dos testes de aceitação efetuados encontram-se no mesmo Apêndice A onde podem ser consultados os requisitos funcionais do sistema e se pode observar que todos os testes de aceitação efetuados foram concluídos com sucesso.

Capítulo 6

Resultados obtidos, conclusões e trabalho futuro

Este capitulo tem como objetivo apresentar a análise do trabalho desenvolvido, que contribui-o para a realização do projeto apresentado. Para isto, vai ser apresentado o conjunto de fases e etapas realizadas ao longo dos dois Semestres de duração do estágio, bem como, a análise de todo o trabalho desenvolvido para a implementação do sistema de distribuição e exibição de conteúdos multiplataforma, e ainda, um conjunto de características a adicionar ao sistema final na eventualidade de um trabalho futuro.

6.1 1º Semestre

Relativamente ao trabalho desenvolvido durante o primeiro Semestre de estágio, estava planeado a definição e documentação de todos as etapas necessárias para iniciar o processo de desenvolvimento do segundo Semestre. Contudo, isto não foi possível devido a dificuldades relacionadas com o objetivo principal do estágio, estudo de *frameworks* para o desenvolvimento multiplataforma. Que, aliadas à conclusão de cadeiras do plano curricular de estudos resultaram em tempo insuficiente para a definição de requisitos e arquitetura necessários para iniciar o processo de desenvolvimento no segundo Semestre.

Este atraso resultou numa reformulação dos objetivos de estágio, tornando a avaliação de *frameworks* de desenvolvimento multiplataforma num objetivo complementar e a construção de um sistema automático de distribuição e exibição de conteúdos multiplataforma no objetivo principal. O que possibilitou a agilização do processo de comparação de tecnologias e permitiu reaproveitar a pesquisa já efetuada tendo em conta o desenvolvimento do referido sistema para as plataformas *Windows*, *Linux*, *Mac OS*, *Android* e *iOS*.

6.2 2º Semestre

Em termos do trabalho desenvolvido durante o segundo semestre, numa primeira fase consistiu na alteração da documentação efetuada até ao momento para refletir as mudanças dos objetivos de estágio e numa segunda fase na especificação e implementação do sistema a desenvolver. Esta segunda fase dividiu-se em cinco etapas fundamentais:

1. **Levantamento de requisitos e definição da arquitetura**
   Nesta etapa, foi feito o levantamento dos requisitos e definição da arquitetura para o sistema de *Digital Signage* multiplataforma. Os requisitos funcionais iniciais foram levantados tendo em conta a integração de clientes multiplataforma com o servidor *Django* fornecido pela empresa *Ubiwhere*. E a definição da arquitetura do sistema foi efetuada com recurso à análise das capacidades oferecidas pelo servidor *Django*.

   Após isto, requisitos funcionais, requisitos não funcionais e arquitetura do sistema foram apresentados à empresa *Ubiwhere* por forma a validar a especificação técnica efetuada e dar inicio à implementação.

2. **Implementação de requisitos**
   Nesta etapa, foram implementados os requisitos definidos na etapa anterior. De forma a possibilitar esta implementação, foi necessário a familiarização com as *frameworks* de desenvolvimento, *Django*, *Electron* e *Cordova*. As quais, sendo completamente novas para o estagiário, necessitaram de um período de adaptação e experimentação, no qual, foi lida a documentação relevante e foram implementadas um conjunto de funcionalidades básicas.

   Com maior conhecimento das tecnologias de desenvolvimento, deu-se inicio a implementação dos requisitos funcionais do sistema a desenvolver. Começando-se por implementar as funcionalidades da aplicação multiplataforma, e posteriormente, integra-las com o servidor *Django* fornecido pela empresa.

   Em termos da integração da aplicação multiplataforma com o servidor *Django* foram tidos alguns problemas na comunicação *Javascript* (linguagem utilizada em *Cordova* e *Electron*) com *Python* (linguagem utilizada em *Django*). E para os resolver, foi necessária a adição de configurações ao servidor *Django* e alteração de alguns aspetos da implementação dos clientes multiplataforma.

   De uma forma geral esta etapa de implementação de requisitos permitiu adquirir bastantes conhecimentos, uma vez que as tecnologias envolvidas nunca tinham sido utilizadas e como tal foi necessário todo um processo de aprendizagem e adaptação desafiante.

3. **Refinamento de requisitos e arquitetura**
   Após finalizar a etapa de implementação anterior, o sistema desenvolvido foi apresentado à empresa

Ubiwhere por forma a perceber se este estava de acordo com o pretendido. Desta apresentação, e a pesar de o sistema apresentar as funcionalidades devidas, surgiu a necessidade de implementar comunicações de alterações em tempo real.

Este facto levou ao refinamento dos requisitos não funcionais do sistema. E, de forma a acomodar comunicação em tempo real, foi feito um estudo da possibilidade de implementação do método *Server-Push* mantendo a arquitetura do sistema já definida. Neste estudo constatou-se que tal era impossível devido ao componente *Django* não suportar as *long living connections* utilizadas em comunicações de tempo real.

Por forma a manter a implementação já efetuada o mais inalterada possível optou-se pela integração no sistema de um componente responsável pelas conexões permanentes. E, após analisar um conjunto de candidatos possíveis, foi escolhido o componente *Meshblu*.

Já com este componente em mente, foram adicionadas as alterações necessárias à arquitetura já definida para que o servidor *Django* pudesse fazer uso de *Push-notifications* como forma de comunicação em tempo real de alterações. O que resultou em mudanças ao nível da implementação do servidor *Django* e das duas aplicações multiplataforma.

Ainda nesta etapa, o componente *Meshblu* foi estudado com recurso à leitura da documentação oficial por forma a facilitar a integração deste componente no sistema na etapa posterior de implementação da nova arquitetura definida.

4. **Alteração da implementação**

   Nesta etapa, e por forma, a implementar a comunicação *Server-Push*, o componente *Meshblu* foi integrado no sistema e foram desenvolvidos os módulos *Changes Module* e *Meshblu Communication Module*.

   Como já referido, com o método de comunicação *Server-Push* o servidor é responsável por enviar dados para os clientes do sistema sem que estes os tenham de pedir especificamente. Com este intuito criou-se o *Changes Module* no componente *Django*. Este módulo é responsável pela criação e envio de notificações para o componente *Meshblu* de forma a que este as propague para os clientes multiplataforma apropriados. O processo de desenvolvimento deste módulo verificou-se ser complexo e demorado pelo facto de o servidor *Django* ser um componente não desenvolvido pelo estagiário e, como tal, requerer aprendizagem sobre a sua implementação e funcionamento.

   Após a integração do *Meshblu* e desenvolvimento do *Changes Module*, passou-se à implementação do módulo responsável pela receção de notificações nas duas aplicações multiplataforma, *Meshblu Commu-*

nication Module. O processo de construção deste módulo foi facilitado devido à utilização de padrões de desenvolvimento em *Javascript*. Que facilitaram a integração de um novo componente nas aplicações multiplataforma mantendo o correto funcionamento da solução já implementada.

5. **Realização de testes e documentação de todas as etapas**
   Nesta última etapa do segundo Semestre foi feita a validação do trabalho desenvolvido com recurso a: testes de compatibilidade, testes arquiteturais, e ainda, testes de aceitação complementares aos efetuados durante todo o processo de implementação. Estes testes permitiram identificar e corrigir alguns problemas na solução desenvolvida, bem como, validar o conjunto de alterações arquiteturais efetuadas para acomodar o método de comunicação *Server-Push*. Após isto, deu-se inicio a uma fase mais ativa na documentação de todas as etapas realizadas.

6.3 Conclusões

Para concluir, o desenvolvimento deste projeto apresentou inúmeras dificuldades relativamente à integração e modificação de componentes não desenvolvidos pelo estagiário, bem como, dificuldades inerentes à aprendizagem do manuseamento de novas tecnologias de desenvolvimento. Apesar disto, e tendo em conta o principal objetivo de criar um sistema de distribuição/exibição de conteúdos multiplataforma, o resultado obtido encontra-se de acordo com o pretendido.

Relativamente às principais tecnologias utilizadas *Electron* e *Cordova*, foi possível perceber que apresentam enormes capacidades e vantagens, no que toca ao desenvolvimento de aplicações para múltiplas plataformas ao eliminar o paradigma de utilização de várias linguagens de programação para suportar diferentes sistemas operativos. No entanto, por serem tecnologia recente, ocorreram mais problemas durante o processo de desenvolvimento e existiu maior dificuldade em encontrar documentação útil à sua resolução. O mesmo não aconteceu com *Django Python*, que por se tratar de uma tecnologia mais madura, muitos dos problemas já tinham sido identificados e documentados. O que se revelou uma enorme vantagem para ultrapassar algumas dificuldades encontradas.

A realização deste estágio permitiu adquirir inúmeros conhecimentos ao nível do desenvolvimento de um projeto real onde sem a utilização de processos de Engenharia não o seria possível terminar em tempo útil. Pois, este projeto necessitou o desenvolvimento de uma solução de raiz (duas aplicações multiplataforma) e a integração da mesma com um componente fornecido (servidor Django), bem como, a alteração de um sistema final já especificado e implementado, para acomodar uma nova necessidade do cliente final, *Ubiwhere*. Todos estes fatores contribuíram para que este projeto se torna-se num desafio enorme em termos de adaptação e obtenção de conhecimentos por parte do estagiário.

6.4 Trabalho Futuro

A prova de conceito apresentada neste projeto foi desenvolvida com o intuito de resolver os problemas de distribuição e exibição de multimédia num caso específico de vendas a retalho (lojas de dispositivos). Para tal, foram criadas duas aplicações multiplataforma que, em conjunto com um servidor central, permitem visualizar conteúdos nas plataformas *Windows*, *Mac OS*, *Linux*, *Android* e *iOS*. Contudo, e apesar de serem suportados os mais populares, pretende-se que futuramente sejam abrangidos mais sistemas operativos, de forma a atingir dispositivos diferentes.

Em qualquer projeto de *Software*, um aspeto chave a ter em conta é a manutenção da solução desenvolvida, de modo a facilitar eventuais resoluções de erros e adição de novas funcionalidades. No entanto, para a prova de conceito neste relatório, a manutenção da solução desenvolvida não foi considerada. Futuramente, e num ambiente de produção, pretende-se que o sistema disponha de atualizações automáticas, para propagar eventuais alterações à implementação do mesmo.

O sistema de distribuição e exibição de conteúdos já descrito neste relatório visa ser uma solução comercializável onde o servidor *Django* é mantido e disponibilizado pela empresa *Ubiwhere*; e as aplicações multiplataforma fornecidas aos seus clientes. Posto isto, e de maneira a proteger a propriedade intelectual presente nestas aplicações, ambiciona-se que, no futuro, sejam utilizadas técnicas automáticas de ofuscação de código para tornar o *Javascript* executado ilegível.

Durante a fase de testes desta prova de conceito, verificou-se que eram dificultadas a observação do estado das notificações enviadas entre os diferentes componentes do sistema e a testabilidade unitária dos módulos implementados. Como forma de resolver estas limitações, pretende-se, no futuro, que o sistema seja provido da capacidade de monitorização de nós (componentes) e que seja possível testar módulos construidos com base na comunicação por eventos.

Relativamente à prova de conceito desenvolvida, apenas permite a exibição de conteúdos fornecidos pelo servidor *Django* e no agendamento imposto pelo mesmo. Futuramente ambiciona-se que o sistema permita: visualizar multimédia local dos dispositivos, definir agendamentos nos dispositivos e sincronizar automaticamente com o servidor central as configurações efetuadas localmente. Por fim, e ainda em termos da exibição de conteúdos, pretende-se que seja possível apresentar multimédia sobreposta e em diferentes partes do ecrã (US.1.18 do Apêndice A).

Em suma, pretende-se enriquecer o sistema desenvolvido em termos da sua testabilidade, manutenção e capacidades de distribuição/exibição de conteúdos multiplaforma.

Bibliografia

[1] 4th Generation Bike Sharing System | Smart Cities and Communities. http://www.ubiwhere.com/en/products/bikeemotion/#.VsIMJXWLRhF. Última visita : 2016-02-15.

[2] 8th - One Effort, Multiple Platforms. http://8th-dev.com/. Última visita : 2016-01-21.

[3] Advanced IDE for iOS; Android - Xamarin Studio. https://www.xamarin.com/studio. Última visita: 2016-01-07.

[4] Apache Cordova. https://cordova.apache.org/. Última visita : 2016-01-21.

[5] Browser Statistics. http://www.w3schools.com/browsers/browsers_stats.asp. Última visita : 2016-02-15.

[6] Build a Native Android UI; iOS UI with Xamarin.Forms - Xamarin. https://www.xamarin.com/forms. Última visita: 2016-01-07.

[7] Category:Language Reference - Xojo Documentation. https://docs.xojo.com/index.php/Category:Language_Reference. Última visita: 2016-01-07.

[8] CLI Reference - Apache Cordova. https://cordova.apache.org/docs/en/latest/reference/cordova-cli/index.html. Última visita: 2016-01-07.

[9] Clipboard · nwjs/nw.js Wiki. https://github.com/nwjs/nw.js/wiki/Clipboard. Última visita: 2016-01-07.

[10] CocoaPods.org. https://cocoapods.org/. Última visita: 2016-01-07.

[11] Code, test, and deploy together with gitlab open source git repo management software — gitlab. https://about.gitlab.com. Última visita: 13-01-2016.

[12] Codename One - Cross-Platform Mobile Development in Java (mobile app development mobile application development). https://www.codenameone.com/. Última visita : 2016-01-21.

[13] Cross-Platform Native Development with Javascript. https://www.nativescript.org/. Última visita : 2016-01-21.

[14] Crossbar.io. http://crossbar.io/. Última visita: 2016-04-07.

[15] Developer megatrends h1 2015. http://www.visionmobile.com/product/developer-megatrends-h1-2015/. Última visita: 13-01-2016.

[16] Device - Apache Cordova. https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-device/index.html. Última visita: 2016-01-07.

[17] Digital Signage | UCVIew. http://www.ucview.com/. Última visita : 2016-02-15.

[18] Digital Signage Software - FrontFace for Public Displays – mirabyte. http://www.mirabyte.com/en/products/frontface-for-public-displays. Última visita : 2016-02-15.

[19] Documentation | Pusher. https://pusher.com/docs. Última visita: 2016-04-07.

[20] DTT signal monitoring platform | Smart Cities and Communities. http://www.ubiwhere.com/en/products/umeter-dtt/#.VsIMJnWLRhF. Última visita : 2016-02-15.

[21] Eclipse - The Eclipse Foundation open source community website. https://eclipse.org/. Última visita: 2016-01-07.

[22] ECMASript 2016 Language Specification. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

[23] Electron. http://electron.atom.io/. Última visita : 2016-01-21.

[24] electron-userland/electron-builder: Complete solution to package and deploy Electron apps. https://github.com/electron-userland/electron-builder. Última visita: 2016-01-07.

[25] electron-userland/electron-packager: Package and distribute your Electron app with OS-specific bundles (.app, .exe etc) via JS or CLI. https://github.com/electron-userland/electron-packager. Última visita: 2016-01-07.

[26] electron/application-packaging.md at master.

[27] Enterprise Mobile App Builder & MBaaS | Appery.io. https://appery.io/. Última visita : 2016-01-21.

[28] EvolveLabs/electron-updater: Cross platform auto-updater for electron apps. https://github.com/evolvelabs/electron-updater. Última visita: 2016-01-07.

[29] FAQ: General | Django documentation | Django. https://docs.djangoproject.com/en/1.9/faq/general/. Última visita: 2016-06-07.

[30] File - Apache Cordova. https://github.com/nwjs/nw.js/wiki/shortcut. Última visita: 2016-01-07.

[31] File System Node.js v6.2.2 Manual; Documentation. https://nodejs.org/api/fs.html. Última visita: 2016-01-07.

[32] File Tranfer - Apache Cordova. https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-file-transfer/index.html. Última visita: 2016-01-07.

[33] Finally, Real-Time Django Is Here: Get Started with Django Channels | Heroku. https://blog.heroku.com/archives/2016/3/17/in_deep_with_django_channels_the_future_of_real_time_apps_in_django. Última visita: 2016-04-07.

[34] Home - EachScape. https://eachscape.com/?nabe=5006181924864000:0. Última visita : 2016-01-21.

[35] Home | Mono. http://www.mono-project.com/. Última visita: 2016-01-07.

[36] https://django-websocket-redis.readthedocs.io/en/latest/introduction.html. Introduction\T1\textemdashdjango-websocket-redis0.4.6documentation. Última visita: 2016-04-07.

[37] Hybrid vs Native Mobile Apps - The Answer is Clear. http://www.ymedialabs.com/hybrid-vs-native-mobile-apps-the-answer-is-clear/. Última visita : 2016-02-22.

[38] Information and Communication Technologies for Travel and Tourism. http://www.ubiwhere.com/en/products/touremotion/#.VsIMJXWLRhF. Última visita : 2016-02-15.

[39] IntelliJ IDEA the Java IDE. https://www.jetbrains.com/idea/. Última visita: 2016-01-07.

[40] ISO/IEC 25010:2011(en). Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en. Última visita: 13-01-2016.

[41] ISO/IEC 9126-1:2001. Software engineering – Product quality – Part 1: Quality model. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749. Última visita: 13-01-2016.

[42] java.com: JAVA + YOU. hhttps://www.java.com. Última visita : 2016-01-21.

[43] Learning JavaScript Design Patterns. https://addyosmani.com/resources/essentialjsdesignpatterns/book/#observerpatternjavascript. Última visita: 2016-01-07.

[44] Learning JavaScript Design Patterns. https://addyosmani.com/resources/essentialjsdesignpatterns/book/#modulepatternjavascript. Última visita: 2016-01-07.

[45] LiveCode. https://livecode.com/. Última visita : 2016-01-21.

[46] Machine-to-machine instant messaging platform for the internet of things. https://github.com/octoblu/meshblu. Última visita: 2016-04-07.

[47] Market share for mobile, browsers, operating systems and search engines | netmarketshare. http://marketshare.hitslink.com/. Última visita: 18-10-2015.

[48] Menu · nwjs/nw.js Wiki. https://github.com/nwjs/nw.js/wiki/menu. Última visita: 2016-01-07.

[49] Message Broker. https://msdn.microsoft.com/en-us/library/ff648849.aspx. Última visita: 2016-07-07.

[50] Metrics and Analytics for ROI in Digital Signage. http://www.ravepubs.com/wp-content/uploads/2013/10/Metrics-and-Analytics-for-ROI-in-Digital-Signage-10-3-13.pptx. Última visita : 2016-02-15.

[51] mirabyte ® Software (International) - Official Homepage. http://www.mirabyte.com/en/. Última visita : 2016-02-15.

[52] Mobile App Development & App Creation Software - Xamarin. https://xamarin.com/. Última visita : 2016-01-21.

[53] Mobile App Development Platform & MBaaS | Appcelerator. http://www.appcelerator.com/. Última visita : 2016-01-21.

[54] MVC – Wikipédia, a enciclopédia livre. https://pt.wikipedia.org/wiki/MVC. Última visita: 2016-06-07.

[55] Native, Cross-Platform Development - Xojo. http://xojo.com/. Última visita : 2016-01-21.

[56] Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options - developer.force.com. https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options. Última visita : 2016-02-22.

[57] Navori Digital Signage Software, Digital Signage Solutions. https://www.navori.com/. Última visita : 2016-02-15.

[58] Node.js. https://nodejs.org/en/. Última visita: 2016-01-07.

[59] Novisign Digital signage software. http://www.novisign.com/software/. Última visita : 2016-02-15.

[60] npm. https://www.npmjs.com/. Última visita: 2016-01-07.

[61] nw-updater. https://www.npmjs.com/package/nw-updater. Última visita: 2016-01-07.

[62] NW.js. http://nwjs.io/. Última visita : 2016-01-21.

[63] nwjs/nw-builder: Lets you build your NW.js apps for mac, win and linux via cli. It will download the prebuilt binaries for a newest version, unpacks it, creates a release folder, create the app.nw file for a specified directory and copies the app.nw file where it belongs. https://github.com/nwjs/nw-builder. Última visita: 2016-01-07.

[64] One framework. - Angular 2. https://angular.io/. Última visita: 2016-01-07.

[65] Overview - redmine. http://www.redmine.org/. Última visita: 13-01-2016.

[66] Push technology - Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Push_technology. Última visita: 2016-04-07.

[67] QT-Home. http://www.qt.io/. Última visita : 2016-01-21.

[68] React Native | A framework for building native apps using React. https://facebook.github.io/react-native/. Última visita : 2016-01-21.

[69] Requests: HTTP for Humans — Requests 2.11.1 documentation. http://docs.python-requests.org/en/master/#. Última visita: 2016-07-07.

[70] Shortcut · nwjs/nw.js Wiki. https://github.com/nwjs/nw.js/wiki/shortcut. Última visita: 2016-01-07.

[71] Signals | Django documentation | Django. https://docs.djangoproject.com/en/1.9/topics/signals/. Última visita: 2016-07-07.

[72] Slack: Be less busy. https://slack.com/. Última visita: 2016-01-07.

[73] Smart Parking Solution | Connected Smart Objects and People. http://www.ubiwhere.com/en/products/uparking/#.VsIMJnWLRhF. Última visita : 2016-02-15.

[74] SQLite Home Page. https://www.sqlite.org/. Última visita: 2016-01-07.

[75] Standards - w3c. https://www.w3.org/standards/. Última visita: 18-01-2016.

[76] Stremio - Watch Instantly. http://www.strem.io/. Última visita: 2016-01-07.

[77] Telerik Mobile App Development Platform, .NET UI Controls, Web, Mobile, Desktop Development Tools. http://www.telerik.com/. Última visita: 2016-01-07.

[78] The State of Native vs. Web vs. Hybrid - DZone Mobile. https://dzone.com/articles/state-native-vs-web-vs-hybrid. Última visita : 2016-02-22.

[79] The WebSocket API. https://www.w3.org/TR/websockets/network-intro.

[80] USER EXPERIENCE WHITE PAPER Bringing clarity to the concept of user experience. http://www.allaboutux.org/files/UX-WhitePaper.pdf. Última visita : 2016-01-21.

[81] vasanthps/nw-test-runner: Test runner for node webkit angular app with mocha and xunit. https://github.com/vasanthps/nw-test-runner. Última visita: 2016-01-07.

[82] viewneo - Easy Digital Signage - PT | viewneo é o premiado sistema de digital signage e intuitiva. https://www.viewneo.com/pt/. Última visita : 2016-02-15.

[83] Visionmobile. http://www.visionmobile.com/. Última visita: 13-01-2016.

[84] Visual Studio - Microsoft Developer Tools. https://www.visualstudio.com/. Última visita: 2016-01-07.

[85] Visual Studio Code - Code Editing. Redefined. https://code.visualstudio.com/. Última visita: 2016-01-07.

[86] websocket - How to build a push system in django? - Stack Overflow. http://stackoverflow.com/questions/10927505/how-to-build-a-push-system-in-django. Última visita: 2016-04-07.

[87] Webstorm: The smartest javascript ide. https://www.jetbrains.com/webstorm/. Última visita: 13-01-2016.

[88] Welcome to NetBeans. https://netbeans.org/. Última visita: 2016-01-07.

[89] Welcome to The Apache Software Foundation! https://github.com/nwjs/nw.js/wiki/shortcut. Última visita: 2016-01-07.

[90] What is loose coupling? - Definition from WhatIs.com. http://searchnetworking.techtarget.com/definition/loose-coupling. Última visita: 2016-06-07.

[91] World Wide Web Consortium (W3C). https://www.w3.org/. Última visita: 2016-07-07.

[92] XMLHttpRequest Standard. https://xhr.spec.whatwg.org/. Última visita: 2016-07-07.

[93] Judith Bishop and Nigel Horspool. Cross-platform development: Software that lasts. *Computer*, 39(10):26–35, October 2006. doi:10.1109/MC.2006.337.

[94] Danyl Bosomworth. Statistics on mobile usage and adoption to inform your mobile marketing strategy. http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/. Última visita: 13-12-2015.

[95] Engin Bozdag, Ali Mesbah, and Arie Van Deursen. A comparison of push and pull techniques for ajax. In *Web Site Evolution, 2007. WSE 2007. 9th IEEE International Workshop on*, pages 15–22. IEEE, 2007.

[96] Andre Charland and Brian Leroux. Mobile application development: web vs. native. *Communications of the ACM*, 54(5):49–53, May 2011. doi:10.1145/1941487.1941504.

[97] Vieri del Bianco, Luigi Lavazza, Sandro Morasca, and Davide Taibi. Quality of open source software: The qualipso trustworthiness model. In *Open Source Ecosystems: Diverse Communities Interacting*, volume 299 of *IFIP Advances in Information and Communication Technology*, pages 199–212. Springer, 2009.

[98] Frans-Willem Duijnhouwer and Chris Widdows. Capgemini open source maturity model. `http://jose-manuel.me/thesis/references/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.3.pdf`, 2013. Última visita : 2016-02-22.

[99] Pedro J. Freire and Rui Ribeiro. Revisão de literatura de frameworks de desenvolvimento móvel multiplataforma. *CAPSI/2013*, 2013.

[100] Bernard Golden. *Succeeding with open source*. Advances in Chemical Engineering 28. Addison-Wesley Professional, 1 edition, 2001.

[101] Wesley Hales. *HTML5 and JavaScript Web Apps*. O'Reilly Media, Inc. ©2012, 2012. ISBN:1449320511 9781449320515.

[102] Carnegie Mellon Software Engineering Institute. Cmmi para desenvolvimento. `http://www.sei.cmu.edu/library/assets/whitepapers/cmmi-dev_1-2_portuguese.pdf`, 2006. Última visita: 13-01-2016.

[103] Mike Jackson, Steve Crouch, and Rob Baxter. Software evaluation: Tutorial-based assessment. `http://software.ac.uk/sites/default/files/SSI-SoftwareEvaluationTutorial.pdf`, November 2011. Última visita: 13-01-2016.

[104] Christopher Murphy, Richard Clark, Oli Studholme, and Divya Manian. *Beginning HTML5 and CSS3: The Web Evolved*. Apress Berkely, CA, USA ©2012, 2012. ISBN:1430228741 9781430228745.

[105] Outsystems. The definitive guide to choosing your enterprise mobile application architecture. http://www.outsystems.com/offer/html5-native-hybrid-mobile-architecture/, 2015.

[106] Alessandro Pasetti. *Software frameworks and embedded control systems*. Springer-Verlag Berlin, Heidelberg ©2002, 2002. ISBN:3-540-43189-6.

[107] Ezequiel Douglas Prezotto and Bruno Batista Boniati. Estudo de frameworks multiplataforma para desenvolvimento de aplicações mobile híbridas. Anais do EATI - Encontro Anual de Tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação.

[108] Ioannis Samolada, Diomidis Spinellis Georgios Gousios, and Ioannis Stamelos. The sqo-oss quality model: Measurement based open source software evaluation. In *Open Source Development, Communities and Quality*, volume 275 of *IFIP – The International Federation for Information Processing*, pages 237–248. Springer, 2008.

[109] Jimmy Schaeffler. *Digital signage: software, networks, adverhttps://www.navori.com/tising, and displays: a primer for understanding the business.* CRC Press, 2012.

[110] Ken Schwaber and Jeff Sutherland. The scrum guide. `http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf`, 2013. Última visita: 13-01-2016.

[111] Abdullahi Mohamud Sharif and Shuib Basri. A study on risk assessment for small and medium software development projects. *International Journal on New Computer Architectures and Their Applications (IJNCAA) The Society of Digital Information and Wireless Communications*, pages 325–335, 2011. ISSN: 2220-9085.

[112] Won Jun Sung, Ji Hyeok Kim, and Sung Yul Rhew. A quality model for open source software selection. In *Advanced Language Processing and Web Information Technology,International Conference on, Advanced Language Processing and Web Information Technology, International Conference on 2007*, pages 515–519, 2007. doi:10.1109/ALPIT.2007.81.

[113] Denis Verdon and Gary McGraw. Risk analysis in software design. *Journal IEEE Security and Privacy*, 2(4):79–84, July 2004. doi:10.1109/MSP.2004.55.

[114] Emily Wheeler. Integrating digital signage and social media. `http://www.x2omedia.com/wp-content/uploads/X2O_Media_WP_Integrating-Digital_Signage_and_Social_Media_To_Launch.pdf` Última visita : 2016-02-15.

Apêndice A

User Stories

Neste anexo são apresentados os requisitos funcionais especificados na forma de US para o sistema a desenvolver, bem como o resultado dos testes aceitação efetuados à implementação realizada. Todos os requisitos seguem o modelo de apresentação referido na Seção 4.2 e são apresentados de seguida na forma de tabelas.

Numeração	US.1.1	**Teste aceitação:** Passou
Como	dispositivo	
Quero	registar-me no sistema	
Para	receber uma identificação única	
Prioridade	5	
Critérios de aceitação:	• Caso o registo seja efetuado com sucesso, devo receber um conjunto de credenciais único.	

Numeração	US.1.2	**Teste aceitação:** Passou
Como	dispositivo	
Quero	identificar-me perante o sistema	
Para	exibir conteúdos multimédia fornecidos	
Prioridade	5	
Critérios de aceitação:	• O processo de identificação não pode ser completado sem fornecer um conjunto de credenciais válido; • Após a identificação com sucesso no sistema, é possível a exibição de conteúdos fornecidos pelo servidor; • No caso de ser impossível fazer a identificação no sistema, deve ser possível a exibição de conteúdos locais.	

Tabela A.1: *User Stories* Para a aplicação multiplataforma

Numeração	US.1.3	**Teste aceitação:** Passou
Como	dispositivo	
Quero	persistir as credenciais recebidas	
Para	Para não ser necessário um novo registo	
Prioridade	5	
Critérios de aceitação:	• As credenciais recebidas são persistidas localmente; • As credenciais são procuradas localmente, e no caso de inexistência das mesmas é feito um novo registo do cliente.	

Numeração	US.1.4	**Teste aceitação:** Passou
Como	dispositivo	
Quero	receber os *settings* definidos no servidor	
Para	Para configurar o meu estado de funcionamento	
Prioridade	4	
Critérios de aceitação:	• Após receção com sucesso dos "*settings*", estes devem puder ser lidos/interpertados Para posterior configuração de funcionamento.	

Numeração	US.1.5	**Teste aceitação:** Passou
Como	dispositivo	
Quero	persistir *settings* recebidos	
Para	posterior uso *offline*	
Prioridade	2	
Critérios de aceitação:	• È criado um ficheiro local, onde vão ser gravados os *settings* recebidos.	

Numeração	US.1.6	**Teste aceitação:** Passou
Como	dispositivo	
Quero	receber o agendamento de conteúdos	
Para	posterior exibição, nos tempos definidos	
Prioridade	5	
Critérios de aceitação:	• È recebido do servidor, o agendamento ativo Para determinado período de tempo; • No caso de ainda não existir um agendamento definido, dever ser exibido um conteúdo multimédia*default*.	

Numeração	US.1.7	**Teste aceitação:** Passou
Como	dispositivo	
Quero	persistir o agendamento de conteúdos	
Para	Para posterior uso *offline*	
Prioridade	2	
Critérios de aceitação:	• O agendamento recebido é persistido localmente; • Quando é recebido um agendamento diferente do já persistido, este deve se sobrepor ao já guardado.	

Tabela A.2: *User Stories* Para a aplicação multiplataforma (continuação)

Numeração	US.1.8	**Teste aceitação:** Passou
Como	dispositivo	
Quero	receber mudanças de agendamento	
Para	refletir estas mudanças em termos de exibição	
Prioridade	5	
Critérios de aceitação:	• È possível em qualquer momento de funcionamento da aplicação receber alterações de agendamento provenientes do servidor; • Ao receber uma mudança de agendamento, esta deve ser refletida na exibição de conteúdos de "imediato"; • As alterações devem ser refletidas nas informações persistidas localmente	

Numeração	US.1.9	**Teste aceitação:** Passou
Como	dispositivo	
Quero	receber mudanças de conteúdos	
Para	refletir estas mudanças em termos de exibiçãoo	
Prioridade	5	
Critérios de aceitação:	• É possível em qualquer momento de funcionamento da aplicação receber alterações a conteúdos; • Uma alteração de conteúdo, pode não implicar uma alteração de agendamento, tendo assim de ser refletida em termos de exibição, mantendo o agendamento já definido; • O conteúdo referente á mudança, deve ser procurado primeiro localmente e no caso de não existir, é feito o download.	

Numeração	US.1.10	**Teste aceitação:** Passou
Como	dispositivo	
Quero	receber mudanças de *settings*	
Para	refletir estas mudanças em termos de opções de funcionamento	
Prioridade	4	
Critérios de aceitação:	• É possível em qualquer momento de funcionamento da aplicação receber alterações aos *settings* do cliente; • Uma alteração de *settings*, deve ser refletida no cliente, sem que o seu estado de funcionamento seja afetado.	

Numeração	US.1.11	**Teste aceitação:** Passou
Como	dispositivo	
Quero	receber todos os conteúdos multimédia referentes a um agendamento	
Para	ter todos os conteúdos a exibir localmente, Como forma de reagir mais "rápido"a alterações	
Prioridade	4	
Critérios de aceitação:	• Após a receção de um agendamento, os conteúdos referentes ao mesmo são procurados localmente, no caso de não existirem, é feito o download de todos estes, antes do inicio da reprodução; • Os conteúdos recebidos, são sempre mantidos localmente por tempo indefinido.	

Tabela A.3: *User Stories* Para a aplicação multiplataforma (continuação)

Numeração	US.1.12	**Teste aceitação:** Passou
Como	dispositivo	
Quero	persistir conteúdos recebidos	
Para	futuro uso, sem que estes tenham de ser novamente descarregados	
Prioridade	4	
Critérios de aceitação:	• Os conteúdos multimédia são guardados localmente, possibilitando a leitura dos mesmos sem que tenham de ser novamente descarregados; • Após uma alteração de agendamento ou conteúdo, é primeiro verificada a existência do ficheiro local referente.	

Numeração	US.1.13	**Teste aceitação:** Passou
Como	dispositivo	
Quero	exibir conteúdos recebidos	
Para	serem visualizados	
Prioridade	5	
Critérios de aceitação:	• Devem ser exibidos os conteúdos dos ficheiros previamente descarregados; • O conteúdo a apresentar, deve ser o definido no agendamento já recebido; • No caso de o conteúdo a exibir ser um vídeo, este deve respeitar os *settings* definidos Para o cliente (reprodução com/sem som); • No caso dos dispositivos mobile, os conteúdos multimédia devem ser apresentados no modo de exibição "*landscape*";	

Numeração	US.1.14	**Teste aceitação:** Passou
Como	dispositivo	
Quero	controlar as trocas de ficheiros a exibir	
Para	cumprir o agendamento imposto	
Prioridade	5	
Critérios de aceitação:	• O conteúdo em exibição deve respeitar o agendamento imposto pelo servidor; • A troca do conteúdo a exibir, deve ser feita no intervalo definido pelo agendamento recebido; • A troca de conteúdo a exibir, deve respeitar os tempos/efeitos de transição impostos pelo agendamento.	

Numeração	US.1.15	**Teste aceitação:** Passou
Como	dispositivo	
Quero	exibir conteúdos multimédia *offline*	
Para	cumprir o agendamento imposto	
Prioridade	2	
Critérios de aceitação:	• Deve ser possível exibir conteúdos multimédia, em situações de falha de rede; • São exibidos conteúdos locais por *default*, ou conteúdos anteriormente recebidos tendo em conta a sua validação Para apresentação;	

Tabela A.4: *User Stories* Para a aplicação multiplataforma (continuação)

Numeração	US.1.16	**Teste aceitação:** Passou
Como	dispositivo	
Quero	ajustar os conteúdos multimédia ao ecrã disponível	
Para	conseguir visualizar corretamente o conteúdo	
Prioridade	4	
Critérios de aceitação:	• Os conteúdos multimédia devem ser redimensionados Para o tamanho de ecrã disponível, de forma a serem visualizados corretamente; • No caso dos dispositivos mobile, os conteúdos a apresentar devem ser redimensionados tendo em conta o modo de apresentação "*landscape*".	

Numeração	US.1.17	**Teste aceitação:** Passou
Como	dispositivo	
Quero	ler informações persistidas	
Para	atuar sobre elas	
Prioridade	5	
Critérios de aceitação:	• Todas as informações persistidas só devem podem ser lidas pelos criadores das mesmas; • O acesso a informações persistidas só pode ser feito pela aplicação multiplataforma.	

Numeração	US.1.18	**Teste aceitação:** Não implementado
Como	dispositivo	
Quero	apresentar conteúdos em diferentes partes do ecrã	
Para	possibilitar a apresentação de vários conteúdos ao mesmo tempo	
Prioridade	2	
Critérios de aceitação:	• São apresentados dois conteúdos ao mesmo tempo, em diferentes locais do ecrã; • Os conteúdos podem ser sobrepostos.	

Tabela A.5: *User Stories* Para a aplicação multiplataforma (continuação)

Apêndice B

Estrutura *JSON* dos dados retornados pela *REST API* e *Changes Module*

Neste anexo é apresentado a estrutura dos dados retornados no formato *JSON* pela *REST API* do servidor, bem como o exemplo de uma notificação retornada pelo *Changes Module* adicionado ao servidor *Django*.

B.1 JSON representativo de *playlists* e agendamentos atribuídos a determinado dispositivo


```
{
  "data": [
    {
      "id": 4,
      "playlist": [
        {
          "id": 1,
          "file_url": "url para download do ficheiro",
          "tags": [],
          "name": "Hobit",
          "duration_ms": 10000,
          "type": "IMG",
          "beginning": "2016-03-29T18:43:00Z",
          "end": "2016-03-29T19:43:01Z"
        }
```

```
      ],
      "name": "Principal",
      "type": 1,
      "transition_stay_ms": 10,
      "align_top": false,
      "align_left": false,
      "transition_type": 1,
      "transition_begin_ms": 10,
      "transition_end_ms": 100,
      "beginning": "2016-03-05T09:08:31Z",
      "end": "2016-03-29T22:08:33Z",
      "created_on": "2016-03-17T15:08:51.409572Z",
      "updated_on": "2016-03-17T17:27:42.716539Z",
      "priority": 1
    }
  ]
}
```


Dentro do *array data* é retornado o conjunto de *playlists* atribuídas a determinado dispositivo, por cada elemento deste *array* (playlist) são apresentados os ficheiros constituintes com respetivo *url* de download, bem como, o agendamento referente à reprodução de cada lista de ficheiros, entre outras informações.

B.2 JSON representativo dos *settings* atribuídos a determinado dispositivo


```
{
  "data": {
        "token": "abc",
        "name": "Settings 1",
        "video_playback_muted": false,
        "content_check_delay_ms": 120000,
        "playback_sync_delay_ms": 120000,
        "updater_check_delay_ms": 120000,
        "appSettings_check_delay_ms": 120000,
        "enable_offline_mode": true
    }
}
```

Esta estrutura é a que representa o conjunto de parâmetros que vão configurar o funcionamento da aplicação responsável pela exibição de conteúdos.

B.3 Exemplo de uma notificação retornada pelo *Changes Module*


```
{
    data: {
        "id": 4,
        "playlist": [{
            "id": 2, "file_url":"url/001_20150407_hobbit_trailer.mp4",
            "tags": [{"id": 1, "name": "cen"}],
            "name": "OutraCosa", "
            duration_ms": 10000,
            "type": "VID",
            "beginning": "2016-03-01T17:22:30Z",
            "end": "2016-03-01T17:22:31Z"
        }],
        "name": "Principal",
        "type": 1,
        "transition_stay_ms": 10,
        "align_top": false,
        "align_left": false,
        "transition_type": 1,
        "transition_begin_ms": 10,
        "transition_end_ms": 100,
        "beginning": "2016-04-05T09:08:31Z",
        "end": "2017-06-29T22:08:33Z",
        "created_on": "2016-03-17T15:08:51.409572Z",
        "updated_on": "2016-08-20T17:21:25.255816Z",
        "priority": 1
        },
    devices: {
        "acbfe50c-304d-490d-b347-99d7fe60fbbf",
        "cd6157c8-5aa1-46af-beca-1dbf01845e1c"
    },
    fromUuid: "6f462ad1-efe3-477a-a23d-3391569d6a0d",
    type: "playlist_changed"}
```

Apêndice C

Modelo de Dados do *Postgre*

O servidor de *back-end* utiliza uma base de dados relacional *open-source* denominada por *Postgre*. O modelo de dados foi definido utilizando as capacidades de *ORM* da framework *Django*. Onde, cada tabela da base de dados corresponde, a um objeto de *Python* (modelo em *Django*), que descreve campos, tipo de dados, e relações de uma entidade da base de dados.

Na Figura C.1 pode ser observado o modelo entidade-relação, no qual constam as seguintes identidades:

- ***channels_channel***: Guarda as informações relativas às *playlists* existentes no sistema.
- ***channels_channel_content***: Tabela de junção resultante da relação *m-to-m* de *channels_channel* com *channels_content*, que guarda os conteúdos que pertencem a uma determinada *playlist*.
- **channels_content**: Guarda os conteúdos multimédia existentes no sistema.
- **core_endpoint_channels**: Tabela de junção resultante da relação *m-to-m* de *channels_channel* com *core_endpoint*, guardando o conjunto de *channels* (*playlists*), atribuídas a determinado cliente multiplataforma.
- ***core_endpoint***: Guarda as informações relativas ao clientes multiplataforma do sistema.
- ***endpoint_settings_settings***: Guarda os conjuntos de *settings* existentes no sistema.
- ***endpoint_settings_updatersettings***: Guarda informações relativas a *updates* de código da aplicação.
- ***expositors_client***: Guarda informações relativas aos clientes do sistema (dono cadeia de lojas)
- ***expositors_store***: Guarda as informações relativas às lojas de um cliente.
- ***expositors_space***: Guarda os espaços para exposição existentes em cada loja.
- ***expositors_expositor***: Guarda os expositores de dispositivos existentes em cada espaço de uma loja.
- ***auth_user***: Guarda informações relacionadas com os utilizadores da aplicação *web* de *back-end*.

- ***auth_permission***: Guarda o conjunto de permissões que podem ser atribuídas aos utilizadores da aplicação *web* de *back-end*.
- ***auth_user_user_permissions***: Tabela de junção resultante da relação *m-to-m* de *auth_user* com ***auth_permission***, que guarda as permissões atribuídas a determinado utilizador.
- ***auth_user_groups***: Tabela de junção resultante da relação *m-to-m* de *auth_user* com *auth_group*, que guarda a atribuição de grupos a utilizadores da aplicação *web* de *back-end*.
- ***auth_group***: Guarda as informações relativas a grupos de permissões existentes no sistema.
- ***auth_group_permissions***: Tabela de junção resultante da relação *m-to-*m de *auth_group* com *auth_permission*, que guarda as permissões atribuídas a determinado grupo.
- ***filer_file***: Guarda os ficheiros multimédia.

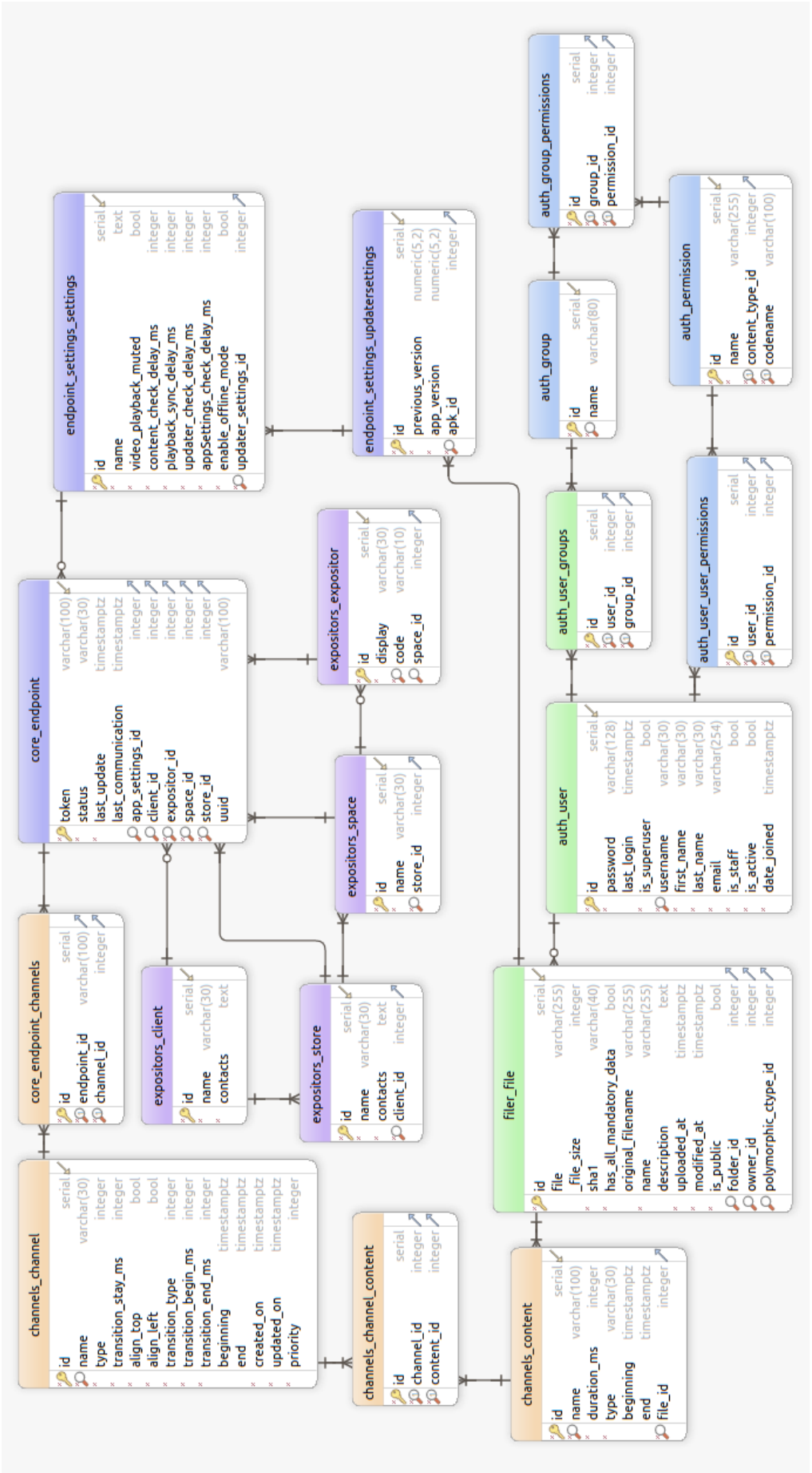


Figura C.1: Diagrama físico da base de dados *Postgres*

Apêndice D

Comunicação entre módulos

Neste Anexo vão ser apresentados as sequências de comunicação entre os módulos constituintes das duas aplicações multiplataforma. Como já referido, estes módulos utilizam um sistema de eventos baseado no padrão Publish/Subscribe como forma de comunicação entre eles.

Inicialmente, quando um módulo é instanciado é feito o registo no *EventBus* dos *listeners* (funções) que devem ser chamados quando é despoletado determinado evento. Posto isto, no seguinte diagrama de sequencia de comunicação são apresentados os eventos que cada módulo subscreve.

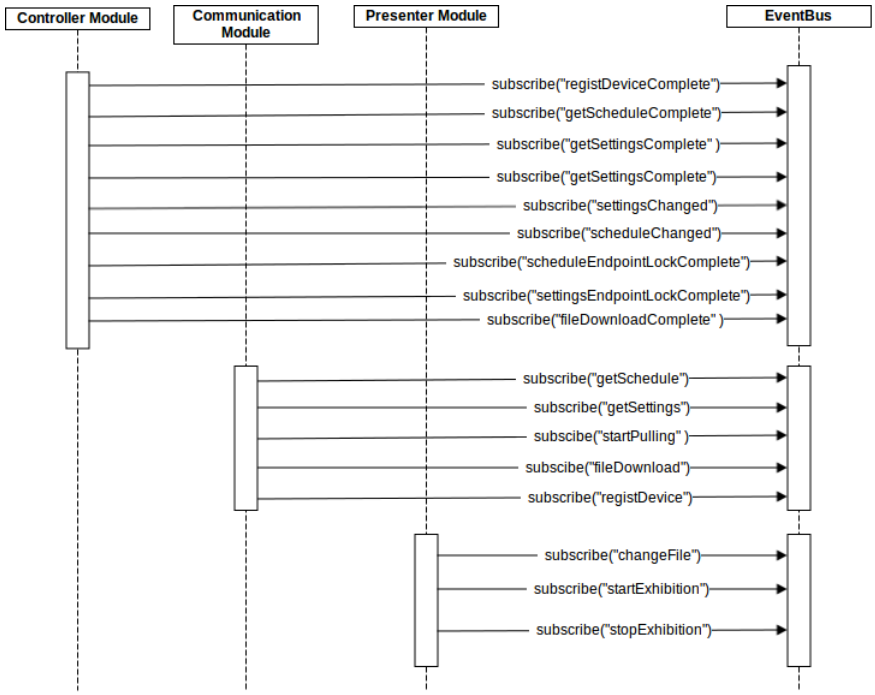


Figura D.1: Diagrama de sequência da comunicação entre módulos (subscrições iniciais).

D.1 *Publish/Fire listeners Phase*

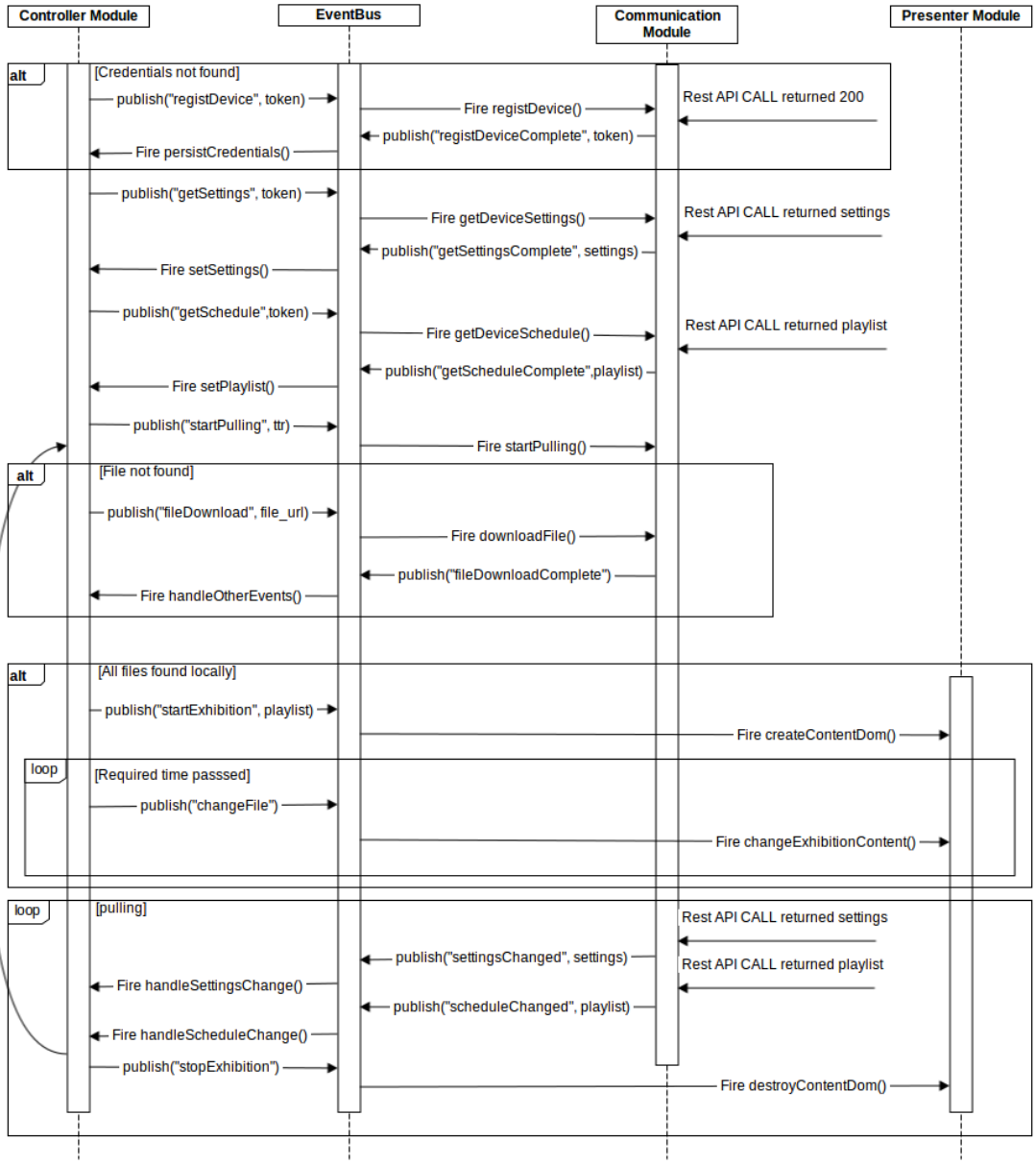


Figura D.2: Diagrama de sequência da comunicação entre módulos (publish/fire).

No diagrama da figura anterior é possivel observar os eventos despoletados por cada módulo (*publish*) assim como as funções (listeners) que são executadas como resultado destes. As sequências alternativas (alt) de eventos que podem ser visualizadas servem para controlar o fluxo de execução interno da aplicação, ocorrendo apenas quando determinada condição é satisfeita. Ainda no diagrama da Figura D.2 é possivel reparar na existência de dois *loops*, o primeiro com a condição *Required time passed* serve para controlar as trocas dos ficheiros a exibir pelo *Presenter Module*. Já o último e mais importante *loop*, representa a propagação de

playlists ou *settings* obtidos durante o *pulling* efetuado entre o *Communication Module* e a *REST API* do servidor. A sequência de mensagens deste *loop* pode ocorrer durante todo o período de funcionamento da aplicação e após se suceda pedido a *REST API* do servidor que retorne um *Status Code* diferente de 304 (*NOT MODIFIED*).

Apêndice E

Sistema de permissões de envio/receção de mensagens do *Meshblu*

Neste Anexo é apresentado em concreto o uso do sistema de permissões do *Meshblu* para promover segurança na troca de mensagens entre os dispositivos do sistema.

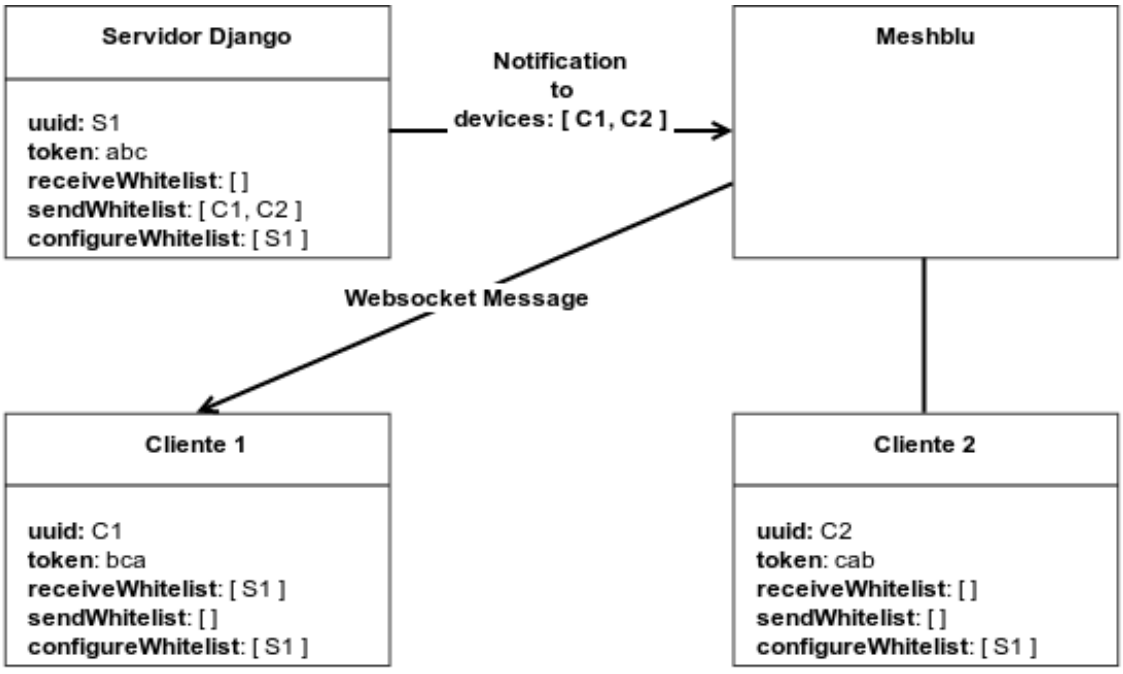


Figura E.1: Envio de uma notificação com a utilização do sistema de permissões do *Meshblu*.

Na Figura E.1 estão representados dois clientes do sistema, cliente 1 e cliente 2, o servidor *Django* e o *Meshblu*. O servidor e clientes que podem ser observados contém os atributos *UUID*, *Token*, *receiveWhitelist*, *sendWhitelist* e *configureWhitelist*. Em concreto, estes atributos têm o seguinte propósito:

- ***UUID*** e ***Token***: identificação única de um dispositivo perante o *Meshblu*;
- ***receiveWhitelist***: lista de *UUIDs* dos quais o dispositivo pode receber mensagens;
- ***sendWhitelist***: lista de *UIIDs* para os quais o dispositivo pode enviar mensagens;
- ***configureWhitelist***: lista de *UIIDs* que podem alterar as configurações do dispositivo.

Nota: os valores dos atributos definidos na figura anterior são meramente exemplos, sendo que *UUIDs* e *Tokens* são atributos de trinta e seis caracteres gerados pelo *Meshblu* a quando do registo do dispositivo.

No caso concreto do envio de uma notificação apresentado na figura anterior, o servidor pretende enviar uma mensagem para o cliente um e dois. É possível observar que a mensagem é enviada inicialmente para o *Meshblu* e é apenas propagada para o cliente um, pois este é o único com permissões definidas para receber mensagens do *UUID S1* (servidor *Django*).

As configurações de dispositivos da Figura E.1 são guardadas na base de dados *Redis* para cada dispositivo registado e apenas podem ser alteradas pelo próprio dispositivo ou pelos *UUIDs* referenciados no atributo *configureWhitelist*.

Foi este sistema de permissões que permitiu definir inequivocamente *Publisher* (Django) e Subscribers (aplicações multiplataforma) das comunicações em tempo real. Permitindo que a comunicação na implementação final apenas possa ser efetuada entre dispositivos devidamente configurados e registados no sistema.

De realçar que, apenas o componente *Django* pode gerir as configurações de permissão atribuídas aos dispositivos registados no sistema.

Apêndice F

Ficheiros de configuração utilizados pelas duas aplicações multiplataforma

Neste Anexo vão ser apresentados os ficheiros *app.config* e *app.credentials*, utilizados para a persistência de dados de configuração relativos ao funcionamento das duas aplicações multiplataforma.

Estes dois ficheiros seguem um formato *JSON*, de forma a facilitar a sua leitura e escrita, e apenas podem ser acedidos pela aplicação que os criou (aplicação multiplataforma) como forma de garantir a segurança dos dados guardados.

F.1 Ficheito app.credentials

```
{
    "token" : "",
    "uuid" : ""
}
```

Neste ficheiro são guardadas as credencias atribuídas ao cliente multiplataforma, de forma a manter a sua identificação constante perante várias execuções da aplicação.

Este ficheiro é criado após o primeiro registo com sucesso do dispositivo no sistema, guardando os atributos de trinta e seis caracteres *token* e *uuid*, gerados pelo *Meshblu*. Mantendo-se inalterado em todas as consequentes execuções da aplicação multiplataforma.

F.2 Ficheito app.config

```
{
    "serverPort" : "porta",
    "serverAdress" : "ip",
    "scheduleEndpoint" : "/api/endpoints/token/schedule/?active",
    "deviceSettingsEndpoint" : "/api/endpoints/token",
    "scheduleLockEndpoint" : "/api/endpoints/token/schedule/lock/",
    "registEnpoint" : "/api/endpoints/register/"
    "meshbluPort": "porta",
    "meshbluAdress": "ip",
    "meshbluWebsocketLocation": "/ws/v2"
}
```

Neste ficheiro os atributos *meshbluPort*, *meshbluAdress*, *serverPort* e *serverAdress* guardam as informações relativas à localização do componente Meshblu e Django. E os restantes atributos, *scheduleEndpoint*, *deviceSettingsEndpoint*, *scheduleLockEndpoint*, *registEnpoint* e *meshbluWebsocketLocation* guardam *endpoints* para os recursos disponibilizados.

Este ficheiro está presenta durante todas as execuções das duas aplicações multiplataforma, e pode ser alterado conforme a alteração da localização ou recursos disponibilizados pelos componentes *Meshblu* ou *Django*.

Apêndice G

Testes às alterações arquiteturais

Neste anexo são apresentadas as medições efetuadas com o *Wireshark* de forma a comparar o número de pacotes trocados nas duas arquiteturas do sistema, tendo em conta o método de comunicação *Client-Pull vs Server-Push*.

As medições aqui apresentadas são relativas ao funcionamento do sistema perante a alteração de conteúdos a exibir pelos clientes multiplataforma e foram obtidas num ambiente local de desenvolvimento devidamente configurado para o efeito. Com todas as interfaces responsáveis pela ligação a redes externas desligadas, de forma a não contribuírem para interferências e criação de *outliers* nos dados obtidos. Posto isto, de seguida são apresentadas as medições obtidas para os testes arquiteturais realizados.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	126661	126661	100.000%	0	0,000%
Between first and last packet	599,171 sec				
Avg. packets/sec	211,394				
Avg. packet size	380,194 bytes				
Bytes	48155714	48155714	100.000%	0	0.000%
Avg. bytes/sec	80370,555				
Avg. MBit/sec	0,643				

Figura G.1: Teste um *Client-Pull*.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	2067	2067	100.000%	0	0,000%
Between first and last packet	577,029 sec				
Avg. packets/sec	3,582				
Avg. packet size	330,006 bytes				
Bytes	682123	682123	100.000%	0	0.000%
Avg. bytes/sec	1182,129				
Avg. MBit/sec	0,009				

Figura G.2: Teste um *Server-Push*.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	14481	14481	100.000%	0	0,000%
Between first and last packet	590,664 sec				
Avg. packets/sec	24,516				
Avg. packet size	373,848 bytes				
Bytes	5413688	5413688	100.000%	0	0.000%
Avg. bytes/sec	9165,426				
Avg. MBit/sec	0,073				

Figura G.3: Teste dois *Client-Pull*.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	2042	2042	100.000%	0	0,000%
Between first and last packet	580,004 sec				
Avg. packets/sec	3,521				
Avg. packet size	328,347 bytes				
Bytes	670485	670485	100.000%	0	0.000%
Avg. bytes/sec	1156,001				
Avg. MBit/sec	0,009				

Figura G.4: Teste dois *Server-Push*.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	6561	6561	100.000%	0	0,000%
Between first and last packet	580,319 sec				
Avg. packets/sec	11,306				
Avg. packet size	389,521 bytes				
Bytes	2555647	2555647	100.000%	0	0.000%
Avg. bytes/sec	4403,864				
Avg. MBit/sec	0,035				

Figura G.5: Teste três *Client-Pull*.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	2067	2067	100.000%	0	0,000%
Between first and last packet	577,029 sec				
Avg. packets/sec	3,582				
Avg. packet size	330,006 bytes				
Bytes	682123	682123	100.000%	0	0.000%
Avg. bytes/sec	1182,129				
Avg. MBit/sec	0,009				

Figura G.6: Teste três *Server-Push*.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	4200	4200	100.000%	0	0,000%
Between first and last packet	584,623 sec				
Avg. packets/sec	7,184				
Avg. packet size	357,682 bytes				
Bytes	1502264	1502264	100.000%	0	0.000%
Avg. bytes/sec	2569,627				
Avg. MBit/sec	0,021				

Figura G.7: Teste três *Client-Pull*.

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	2042	2042	100.000%	0	0,000%
Between first and last packet	580,004 sec				
Avg. packets/sec	3,521				
Avg. packet size	328,347 bytes				
Bytes	670485	670485	100.000%	0	0.000%
Avg. bytes/sec	1156,001				
Avg. MBit/sec	0,009				

Figura G.8: Teste três *Server-Push*.

Nas figuras G.1 a G.8 são apresentados os valores de vários parâmetros avaliados pelos dados obtidos com recurso a ferramenta *Wireshark*. Estes dados são referentes a quatro testes efetuados, com duração de dez minutos cada onde foi variada a frequência de *pulling* da comunicação *Client-Pull* por forma a perceber se esta se podia assemelhar à comunicação *Server-Push* introduzida no sistema.