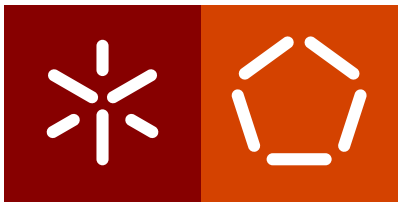
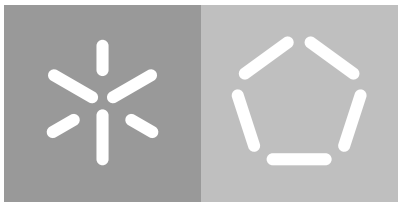


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Victor Manuel Da Cunha

Aplicações para TV baseadas em Templates

December 2019



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Victor Manuel Da Cunha

Aplicações para TV baseadas em Templates

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
António Nestor Ribeiro

December 2019

AGRADECIMENTOS

Gostaria de agradecer às pessoas que me ajudaram na *Altice Labs* de Aveiro, sendo de salientar os supervisores da empresa, Bernardo Cardoso e Hugo Dias, e o Arthur Neves que me ajudou a compreender os *templates* a implementar.
Também quero agradecer ao meu orientador, o professor António Nestor Ribeiro pela sua ajuda durante todo este processo.
Finalmente agradeço à minha família por todo o suporte que me deram.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração. Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

RESUMO

A empresa Altice Labs está a desenvolver uma nova plataforma Internet Protocol television (IPTV) para vir substituir aquela que hoje possui, sendo esta assente em tecnologias standards e normas abertas permitindo uma maior evolução e resiliência. Para verificar se esta é capaz de incorporar serviços usados em produção, a empresa decidiu implementar um desses serviços na nova plataforma. O serviço escolhido foi um serviço *backoffice*, designado por Landing Pages (LP) que permite a criação de aplicações baseadas em configuração de *templates* pré-definidos. As aplicações são apresentadas através de um interpretador presente no cliente, como nas suas set-top boxes. Esta dissertação de mestrado tem como objetivo a implementação deste interpretador das aplicações *Landing Pages (LP)* na nova plataforma, e também verificar se esta é capaz de suportar os serviços que são atualmente disponibilizados. A implementação foi feita em JavaScript tendo sido usado o *Google Chrome* como *web browser* e a plataforma NW.js (node-webkit) como ambientes de testes, visto que a especificação da set-top box, onde a plataforma deverá correr, ainda é desconhecida.

Uma análise do serviço e dos seus *templates* levou à divisão desses em grupos. Cada grupo foi depois implementado sequencialmente num processo iterativo e incremental. A implementação revelou que a plataforma é capaz de incorporar as funcionalidades pretendidas mas apresenta falhas que devem ser corrigidas. Nesta dissertação apresenta-se o processo seguido para recriar todos os *templates* na nova plataforma e correspondente análise das suas funcionalidades na nova plataforma. Este processo forneceu indicações sobre a forma como o interpretador e a plataforma deverão evoluir para permitir a implementação dos serviços que a Altice Labs pretende ver disponibilizados.

Palavras-Chave: plataforma *Internet Protocol television (IPTV)*, desenvolvimento de aplicações, *templates*.

ABSTRACT

Altice Labs is developing a new Internet Protocol television (IPTV) platform to replace the one it currently possesses, being based on standard technologies and open standards allowing for greater evolution and resilience. To check if it is capable of incorporating services used in production, the company decided to implement one of these services on the new platform. The chosen service was a back office service, designated by Landing Pages (LP) that allow the creation of applications based on predefined templates. Applications are then presented through an interpreter present on the client, as in their set-top boxes. This master's dissertation aims to implement this interpreter of LP applications on the new platform, and also check if it is able to support the services that are currently available. The implementation was made in JavaScript using *Google Chrome* as a web browser and the *NW.js* platform (node-webkit) as tests environments, since the specification of the set-top box, where the platform should run, is still unknown.

An analysis of the service and its templates led to their division into groups. Each group was then implemented sequentially in an iterative and incremental process. The implementation revealed that the platform is capable of incorporating the intended features but it has flaws that must be corrected. This dissertation presents the process followed to recreate all templates on the new platform and corresponding analysis of its features on the new platform. This process provides pointers on how the interpreter and platform should evolve to enable deployment of the services that the company Altice Labs intends to make available.

Keywords: IPTV platform, applications, templates.

CONTENTS

1 INTRODUÇÃO 1
- 1.1 Contexto 2
- 1.2 Problema 3
- 1.3 Desafios 3
- 1.4 Objetivos 3
- 1.5 Estrutura da dissertação 4

2 ESTADO DA ARTE 6
- 2.1 Uma visão global sobre IPTV 6
- 2.2 Soluções IPTV 7
  - 2.2.1 Solução IPTV do Open IPTV Forum 8
  - 2.2.2 Nagra OpenTV 11
  - 2.2.3 MediaKind Mediaroom 14
- 2.3 Desenvolvimento de aplicações para televisão 16
  - 2.3.1 Desenvolvimento de *User Interface (UI)* para a Samsung 16
- 2.4 Sumário 18

3 ANÁLISE E ABORDAGEM DA PLATAFORMA LANDING PAGES 19
- 3.1 Plataforma Landing Pages 19
  - 3.1.1 Arquitetura Plataforma Landing Pages 20
  - 3.1.2 Aplicação Landing Pages 21
- 3.2 Grupos de templates da plataforma Landing Pages 23
  - 3.2.1 Grupo 1 23
  - 3.2.2 Grupo 2 24
  - 3.2.3 Grupo 3 25
  - 3.2.4 Grupo 4 26
  - 3.2.5 Grupo 5 28
  - 3.2.6 Grupo 6 29
  - 3.2.7 Síntese 31
- 3.3 Plataforma de desenvolvimento 33
  - 3.3.1 Criação de *User Interface* 33
  - 3.3.2 Navegação entre *screens* 34
- 3.4 Criação do interpretador 34
- 3.5 Sumário 35

4 DESENVOLVIMENTO 36

4.1 Problemas e Decisões 36
4.1.1 Motor de animação 37
4.1.2 Motor de Escala 37
4.1.3 Gestão dos elementos visuais 38
4.1.4 Alterações devido a atualizações 38
4.1.5 Falta de documentação 38
4.1.6 Navegação entre screens 39
4.1.7 Sumário 39
4.2 Implementação 39
4.2.1 Arquitetura do módulo 40
4.2.2 Comunicação com a API 42
4.2.3 Navegação no router 43
4.2.4 Screens e Views 43
4.2.5 Navegação nas screens 52
4.3 Sumário 52
5 CASOS DE ESTUDO E RESULTADOS 54
5.1 Ambiente de execução 54
5.2 Casos de estudo 54
5.2.1 Primeira Aplicação 55
5.2.2 Segunda Aplicação 59
5.3 Discussão dos resultados obtidos 62
5.3.1 Divergências nas animações 62
5.3.2 Funcionalidades não implementadas 63
5.4 Sumário 64
6 CONCLUSÃO E TRABALHO FUTURO 65
6.1 Conclusão 65
6.2 Trabalho futuro 66

A CONFIGURAÇÃO PÁGINA INICIAL DA APLICAÇÃO 1 71
B CONFIGURAÇÃO PÁGINA INICIAL DA APLICAÇÃO 2 73

LIST OF FIGURES

Figura 1 Modelo em camada que mostra as relações entre os dominios e camadas do modelo OSI [ETSI TS 102 034][9] 8
Figura 2 Âmbito da solução *Open IPTV Forum (OIPF)* IPTV [11] 9
Figura 3 Arquitetura teórica de um sistema para *OpenTV Platform*, segundo *Nagra* .[24] 13
Figura 4 Arquitetura da plataforma Microsoft TV IPTV Edition [29] 15
Figura 5 Arquitetura simplificada da plataforma Landing Pages 20
Figura 6 Exemplo de um *template* com um *background* e um menu 21
Figura 7 Árvore de configuração de uma app 22
Figura 8 Template do grupo 1 23
Figura 9 Template do grupo 2 24
Figura 10 Primeiro template do grupo 3 25
Figura 11 Segundo template, variação do primeiro template do grupo 3 26
Figura 12 Primeiro template do grupo 4 27
Figura 13 Segundo template do grupo 4 27
Figura 14 Primeiro template do grupo 5 28
Figura 15 Segundo Template do grupo 5 29
Figura 16 Primeiro template do grupo 6 30
Figura 17 Segundo template do grupo 6 31
Figura 18 Os módulos são baixados e carregados localmente 33
Figura 19 Diagrama de classes do modulo 41
Figura 20 Elementos do Grupo 1 47
Figura 21 Elementos do Grupo 2 48
Figura 22 Elementos do Grupo 3 48
Figura 23 Elementos do Grupo 4 49
Figura 24 Elementos do Grupo 5 50
Figura 25 Elementos do Grupo 6 51
Figura 26 Página inicial da aplicação na nova plataforma 55
Figura 27 Página inicial da aplicação na plataforma hoje em uso e na nova plataforma 56
Figura 28 Página de conteúdo da aplicação 57
Figura 29 Página de conteúdo da aplicação na plataforma em uso e na nova plataforma 58

Figura 30	Página de conteúdo da aplicação modificada	59
Figura 31	Página inicial da aplicação na nova plataforma	60
Figura 32	Página inicial da aplicação na plataforma em uso	60
Figura 33	Página de conteúdo da aplicação	61
Figura 34	Página de conteúdo modificada	62

LIST OF TABLES

Tabela 1	Lista dos elementos e das suas configurações	32
Tabela 2	Elementos identificados e a implementação efetuada	45

ACRONYMS

A

ADK Application Development Kit.

ANACOM Autoridade Nacional de Comunicações.

API Application programming interface.

C

CA Acesso conditional.

CAS Sistema de acesso condicional.

D

DAE Declarative Application Environment.

DASH Dynamic Adaptive Streaming over HTTP.

DIT Digital, Internet e Televisão.

DLNA Digital Living Network Alliance.

DOM Document Object Model.

DRM Digital Right Management.

DTV Digital Television.

DVB-T Digital Video Broadcasting — Terrestrial.

DVR Digital Video Recording.

E

EAS Sistema de alerta de emergência.

EPG Electronic Program Guide.

G

GEM Globally Executable Multimedia Home Platform.

H

HBBTV Hybrid broadcast broadband TV.

HD High Definition.

HTML HyperText Markup Language.

I

ICC Instant Channel Change.

IP Internet Protocol.

IPTV Internet Protocol television.

ITU FG IPTV International Telecommunication Union focus group on IPTV.

ITV Televisão Interativa.

J

JSON JavaScript Object Notation.

L

LP Landing Pages.

M

MPEG Moving Picture Experts Group.

MPEG-2 *Moving Picture Experts Group (MPEG)* norma 2.

MPEG-4 MPEG norma 4.

MVC Model View Controller.

N

NMP Nagra Media Player.

NTS network Time Shifting.

O

OIPF Open IPTV Forum.

OITF Open IPTV Terminal Function.

OSI Open System Interconnection.

OTT Over the Top.

P

PAE Procedural Application Environment.

PF Presentation Framework.

PIP picture-in-picture.

Q

QOE Quality of Experience.

QOS Quality of Service.

R

RSS Rich Site Summary.

S

SAP Second audio program ou Segundo programa de aúdio.

SD Standard definition.

STB Set Top Box.

SVG Scalable Vector Graphics.

T

TDT Televisão digital terrestre.

TV televisão.

U

UI User Interface.

V

VOD Video On Demand.

X

XML eXtensible Markup Language.

1

INTRODUÇÃO

A *televisão (TV)* transformou-se nos últimos 20 anos numa plataforma interativa, onde o utilizador pode interagir com a TV de maneira a obter informações ou personalizar a sua experiência, designando-se como *Televisão Interativa (iTV)*. Esta transformação é mais aparente com o aparecimento de aplicações dita interativas, como o *Electronic Program Guide (EPG)* e *Digital Video Recording (DVR)*, que são usadas quotidianamente. Nesta evolução a TV também ganhou acesso à internet e a novas aplicações que tiveram de ser redesenhadas de maneira a funcionar neste ambiente com *inputs* restritos.

A integração da internet com a televisão, permitiu a distribuição de programas de televisão através de redes *Internet Protocol (IP)*, designado por Internet Protocol television (IPTV), o que permitiu a criação de novos serviços como *network Time Shifting (nTS)*, que permite a um utilizador ver programas de televisão atrasados relativamente a sua programação normal.

Com esta evolução e com o aparecimento de dispositivos móveis, cada vez mais usados, os operadores de TV tiveram de mudar as suas ofertas para manter e atrair clientes, passando de transmissor de canais de televisão para fornecedores de diversos serviços, como *Video On Demand (VOD)*, *streaming*, gravações na *cloud*, nTS, subscrições e aplicações interativas. Estes serviços são acessíveis a partir da TV através de aplicações interativas, sendo essas geralmente acessíveis unicamente a partir da *Set Top Box (STB)* fornecida pelo operador. A *Autoridade Nacional de Comunicações (ANACOM)* define a STB como "um equipamento descodificador que se liga ao televisor e a uma fonte externa de sinal (cabo *ethernet*, cabo coaxial, linha telefônica ou antena tradicional) e transforma esse sinal de forma a que a emissão possa ser vista no televisor" [2]. Para além disto, estas também fornecem serviços como DVR e permitem aceder a uma gama de aplicações interativas, disponibilizadas pelo operador. As principais funcionalidades duma STB são [17]:

- descodificar o sinal de entrada;
- verificar os direitos de acesso e os níveis de segurança;
- apresentar o conteúdo na TV;
- processar e apresentar serviços iTV e da Internet.

A STB é, assim, um mini-computador, geralmente conectado à internet, que adiciona capacidades ao dispositivo a que está ligado.

As aplicações interativas devem correr em diferentes STB, com diferente *hardware* e *software*, e possivelmente outros dispositivos como televisões inteligentes. De maneira a facilitar a criação e o porte de aplicações para diferentes plataformas é estudado uma plataforma de criação de aplicações baseadas em *templates* desenvolvida pela *Altice Labs* fazendo o porte dessa para uma nova plataforma.

1.1 CONTEXTO

A Altice Labs, e mais precisamente a sua direção *Digital, Internet e Televisão (DIT)*, é responsável pela conceção, desenvolvimento e teste de aplicações e plataformas para TV, web e dispositivos móveis, que são usadas dentro e fora do grupo Altice, tendo como exemplo as aplicações interativas do serviço MEO IPTV e também as aplicações MEO Go e MEO Remote. Dentro desta direção, o grupo da Televisão Interativa (iTV) está focado na manutenção da plataforma MEO IPTV, na criação de novas funcionalidades e no estudo de plataformas alternativas. Hoje em dia, o serviço de televisão MEO IPTV está assente na plataforma propriedade da MediaKind[1], chamada Mediaroom, para distribuição do seu serviço, sendo o seu cliente instalado em mais de 40 milhões de STB em todo o mundo [6]. Esta plataforma, por ser propriedade de outros, tem uma arquitetura definida e limitações próprias, que a Altice Labs não pode modificar. O grupo iTV da *Altice Labs*, durante vários anos, estudou a plataforma, usou-a e explorou-a até os seus limites, personalizando quase todas as funcionalidades base, mas, sendo uma plataforma fechada, é cada vez mais difícil criar aplicações inovadoras e distintivas, estando-se limitado ao que a plataforma oferece. Assim, o grupo Altice está a desenvolver uma plataforma que espera poder ser usada em todas as suas operações, que forneceria as mesmas funcionalidades hoje disponíveis, e que seria assente em tecnologias *standards* e normas abertas que permitam uma maior evolução e resiliência. Esta nova plataforma de desenvolvimento, está a ser construída usando essencialmente JavaScript, CSS3 e HTML5, que são normas da web e tecnologias com anos de uso e uma comunidade ativa, resultando numa evolução constante destas. Além disto, estas tecnologias são hoje usadas em outras plataformas IPTVs como *Hybrid broadcast broadband TV (HbbTV)*, um *standard* da indústria para a normalização da transmissão de TV, serviços IPTV e acesso a internet para os consumidores através de *smart TVs* e STB.

1 https://www.mediakind.com/pt/

1.2 PROBLEMA

A Altice Labs, já possui uma versão inicial da nova plataforma, que segue uma estrutura modular que permite a adição de funcionalidades na forma de módulos, sendo necessário testá-la, verificando se esta é capaz de incorporar funcionalidades hoje disponibilizadas e se é compatível com serviços de *backend* usados em produção. A empresa possui um *backoffice* designado por Landing Pages (LP) que permite escolher *templates* de aplicações pré-definidas, configurá-los e criar uma aplicação interativa para TV pronta a ser distribuída, sem necessidade de programar. As aplicações criadas no *backoffice* são depois disponibilizadas através de uma *Web* Application programming interface (API) usado pelo módulo das LP responsável por, a partir da configuração, recriar a UI e a lógica das aplicações na STB. Assim, é necessário criar e incorporar um módulo na nova plataforma que irá recriar as aplicações LP, através das suas configurações.

1.3 DESAFIOS

A nova plataforma da Altice deverá correr numa STB mas a especificação desta é desconhecida, sendo assim difícil testar e avaliar o desempenho real da plataforma e das aplicações recreadas, por isso, será usado um *web browser* e a plataforma *NW.js* como ambientes de testes. A criação e integração de um módulo numa plataforma privada em desenvolvimento e com documentação limitada é o maior desafio desta dissertação, uma vez que este módulo deve servir como interpretador de aplicações interativas LP e ser escrito em JavaScript.

1.4 OBJETIVOS

A dissertação possui dois objetivos, que são:

- o desenvolvimento de um interpretador para as aplicações LP incorporado na nova plataforma da Altice, na forma de um módulo da plataforma e desenvolvido usando JavaScript;
- a avaliação da capacidade da plataforma para incorporar serviços da empresa usados em produção.

O primeiro objetivo pode ser dividido em subobjetivos que refletem o número de *templates* que deverão ser interpretados. Antes de os listar, é necessário definir o que é um *template* e uma aplicação LP, assim como os grupos de *templates* que serão implementados. Um *template* é aqui definido como um plano construído, onde são especificados os componentes visuais, as suas posições, ações que podem realizar e fontes de dados usados por estes. Cada *template* possui um conjunto de componentes predefinidos, sendo agrupados segundo os

seus usos e as relações entre eles. Uma aplicação LP é uma aplicação interativa construída usando um conjunto de *templates* relacionados, usados em conjunto para a apresentação de conteúdo. A aplicação é composta por uma página de entrada, designada por *home page*, e um conjunto de outros *templates* relacionados com essa. Cada *template* oferece uma gama de configuração e comportamento diferente. A partir da *home page* são definidos ligações para outros *templates* usados na aplicação. Atualmente existem dez *templates* diferentes na plataforma, agrupados em seis grupos, sendo os critérios de agrupamento a lógica presente e como podem ser usados na construção de uma aplicação LP, isto é, excluindo um grupo que contem *templates* que podem ser usados em mais do que um grupo, cada grupo permite a criação de uma aplicação LP. Os grupos são apresentados na secção 3.2.
Assim, os subobjetivos são:

- O interpretador deve permitir a visualização e o uso das aplicações criadas usando os *templates* do grupo 1;
- O interpretador deve permitir a visualização e o uso das aplicações criadas usando os *templates* do grupo 2;
- O interpretador deve permitir a visualização e o uso das aplicações criadas usando os *templates* do grupo 3;
- O interpretador deve permitir a visualização e o uso das aplicações criadas usando os *templates* do grupo 4;
- O interpretador deve permitir a visualização e o uso das aplicações criadas usando os *templates* do grupo 5;
- O interpretador deve permitir a visualização e o uso das aplicações criadas usando os *templates* do grupo 6.

Uma apresentação mais detalhada de cada grupo de *templates* da plataforma LP será feito mais adiante nesta dissertação. Além disto, será feito um estudo da plataforma LP e da sua *Web API*, de maneira a perceber quais os seus objetivos, a sua arquitetura, os diferentes *templates* e o formato das configurações guardadas.

1.5 Estrutura da dissertação

A dissertação é dividida em seis capítulos.

O primeiro apresenta o contexto, problema, desafios e objetivos da dissertação.

O segundo capítulo, *Estado da arte* foca-se nas tecnologias IPTV em uso, e apresenta uma definição para IPTV. São apresentadas nesse capítulo três soluções *IPTV* que são o *Open IPTV Forum IPTV Solution*, o *Nagra OpenTV* e a plataforma *MediaKind Mediaroom*. Para além

disso, é apresentado o método de desenvolvimento de aplicações para televisão aconselhado pela *Samsung*.

O terceiro capítulo, faz a análise da plataforma LP e descreve a sua arquitetura e os seus *templates* para aplicações. Nesse capítulo, também é feito a análise da plataforma de desenvolvimento, explicando como são criadas aplicações, sendo sugeridos passos para o desenvolvimento do interpretador a partir dessa análise.

O quarto capítulo, descreve como o desenvolvimento do interpretador foi feito. O capítulo é dividido em duas partes. Na primeira parte é feito a análise dos problemas encontrados e as decisões tomadas para os resolver, enquanto na segunda parte foca-se a implementação realizada com a apresentação da arquitetura do interpretador criado.

O quinto capítulo, apresenta duas aplicações (simples e complexa) que foram criadas e reproduzidas na nova plataforma, com a análise dos resultados obtidos comparando o aspeto e as funcionalidades das aplicações na nova plataforma e na plataforma hoje em uso.

No sexto capítulo, conclui-se a dissertação apresentando as conclusões tiradas do trabalho efetuado e os trabalhos futuros que deverão ser efetuados quer no módulo criado quer na plataforma.

2

ESTADO DA ARTE

Neste capítulo é apresentado uma visão global sobre IPTV, onde é explicado o que é IPTV, os seus desafios e é apresentado uma definição para IPTV. Depois, é definido o que constitui uma solução IPTV, explicando os domínios que engloba e, por consequência, os problemas que deve resolver. O capítulo foca-se em soluções IPTVs em utilização, os serviços que propõem, a sua organização e como as aplicações interativas são criadas e distribuídas, sendo depois analisado o desenvolvimento de aplicações neste domínio.

2.1 UMA VISÃO GLOBAL SOBRE IPTV

O IPTV é uma forma de televisão digital, ou *Digital Television (DTV)*, onde, contrariamente à televisão tradicional analógica, a transmissão e recepção de imagem e som é feita através de sinais digitais, sinais codificados de forma binária. Isto permitiu aumentar o número de canais, a qualidade de recepção e a resolução desses, para além de possibilitar o uso de um canal de comunicação bidirecional entre o utilizador e o operador, possibilitando a criação de aplicações interativas, teletexto, EPG, a transmissão de canais de rádios, e mais.

Em Portugal, a DTV mais conhecida é a *Televisão digital terrestre (TDT)* e usa uma largura de banda de 8MHz, sendo implementada usando a norma *Digital Video Broadcasting — Terrestrial (DVB-T)*. Esta norma define como canais de televisão e outros conteúdos multimédias são codificados e transmitidos ao cliente. A codificação de imagens e áudio é feito usando a norma *MPEG norma 2 (MPEG-2)*, uma norma de compressão definida como uma *"codificação genérica para imagens em movimento e informação de áudio associado"*, para canais *Standard definition (SD)*, sendo substituída pela norma *MPEG norma 4 (MPEG-4)* quando é necessário um maior nível de compressão, como acontece em canais *High Definition (HD)* [28].

Sendo uma forma de DTV, IPTV lida com abordagens, tecnologias e protocolos para a distribuição de conteúdo SD e HD, em tempo real e *on-demand*, através de redes baseadas no protocolo IP. IPTV alcança todos os pré-requisitos de *Quality of Service (QoS)*, *Quality of Experience (QoE)*, e *Acesso conditional (CA)* (segurança). Da mesma maneira, preenche os requisitos para a gestão de *blackout*, que permite limitar o acesso a conteúdos de um canal a certos utilizadores, e o *Sistema de alerta de emergência (EAS)*. Também possibilita a

implementação de serviços de legenda, controlo parental, recolha de dados para o sistema de medição de audiência, *Second audio program ou Segundo programa de aúdio (SAP)* para a transmissão e recepção no cliente de diversos canais de áudios, permitindo a transmissão do áudio em diversas línguas, e *picture-in-picture (PiP)* [19]. Para além disto, IPTV suporta todos os requisitos de negócio, como a distribuição de serviços, faturação, aprovisionamento, e requisitos de proteção de conteúdo com, por exemplo, *Sistema de acesso condicional (CAS)* ou *Digital Right Management (DRM)*, muitas vezes associados com a distribuição de conteúdos comerciais [19]. O conteúdo transmitido através de IPTV pode ser entregue para qualquer dispositivo compatível com acesso à internet, permitindo atingir novos consumidores e disponibilizar mais serviços. Uma ligação *broadband* é usada como meio de transmissão, e a definição oficial para IPTV, aprovada pelo *International Telecommunication Union focus group on IPTV (ITU FG IPTV)*, é :
" IPTV é definido como sendo serviços multimédias como televisão /áudio /texto /*graphics* /dados transmitidos através duma rede baseada no protocolo IP, para fornecer os níveis de QoS e QoE, segurança, interatividade e confiabilidade" [16].

2.2 SOLUÇÕES IPTV

Uma solução IPTV é aqui considerada como uma plataforma desenvolvida por fabricantes de software ou hardware, que pode ser implementada por um fornecedor de serviço, como empresas de telecomunicações ou de televisão por cabo, para oferecer serviços IPTV. Estas soluções especificam como o conteúdo é adquirido, tratado e apresentado ao cliente. Para facilitar a interoperabilidade das soluções, a maioria deles especificam um *middleware* próprio que permite a qualquer dispositivo com este *middleware* de aceder aos serviços da plataforma.

Uma solução IPTV engloba diferentes domínios, envolvendo diferentes entidades, apresentados na Figura 1. Estes domínios são definidos como [9]:

- o fornecedor de conteúdo, que possui ou está licenciado para vender conteúdo ou ativos e fornece os metadados descritivos;
- o fornecedor de serviço, que fornece o serviço IPTV ao cliente, utilizando conteúdo adquirido ou licenciado dos fornecedores de conteúdo;
- o fornecedor de acesso a rede IP, que fornece ao cliente uma conexão ao fornecedor de serviço. O sistema de entrega é geralmente constituído por uma rede de acesso e núcleo ou redes de *backbone*. A rede de entrega é transparente para o tráfego IP, podendo vir a ser relevante para real-time *streaming*. Esta rede é controlada por um fornecedor de rede;

- a *Home*, que corresponde ao domínio onde os serviços são consumidos. Neste domínio podem existir um ou mais dispositivos IPTV, geralmente uma STB fornecida pelo fornecedor de serviço.

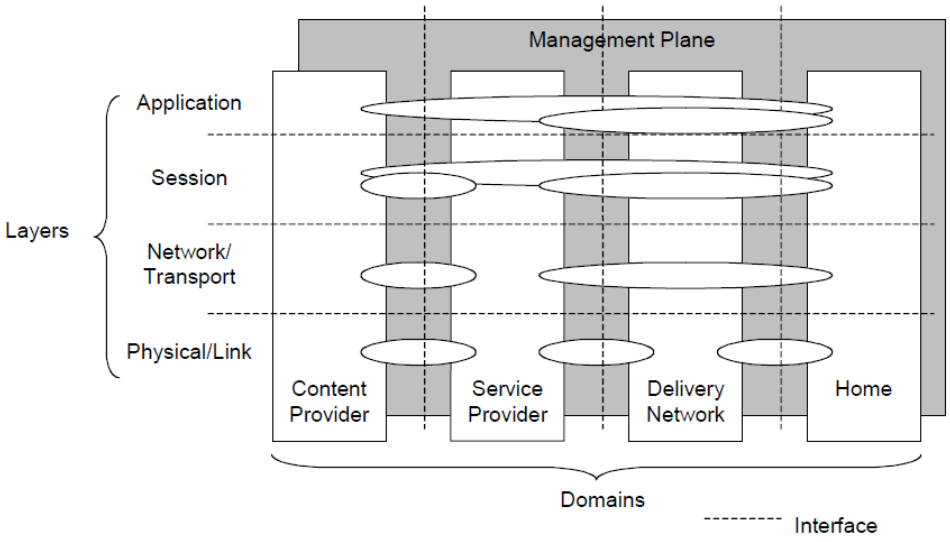


Figura 1: Modelo em camada que mostra as relações entre os dominios e camadas do modelo OSI [ETSI TS 102 034][9]

A Figura 1 apresenta as conectividades existentes entre os domínios segundo o modelo *Open System Interconnection (OSI)*. Na figura é possível verificar que o fornecedor de conteúdo comunica com os aparelhos do domínio *Home* ao nível das aplicações e de sessão, contudo ao nível físico este só é ligado ao seu fornecedor de acesso. A seguir são apresentadas algumas soluções que os fornecedores de serviços podem implementar, especificando a sua arquitetura e os serviços que oferecem.

2.2.1 *Solução IPTV do Open IPTV Forum*

O Open IPTV Forum (OIPF) é um consórcio e uma organização de normalização cujo objetivo é definir e publicar normas para a arquitetura de soluções IPTV fim-a-fim. Em 2014, o OIPF integrou a associação Hybrid broadcast broadband TV (HbbTV). Em janeiro de 2014, o OIPF publicou a versão 2.3 da sua solução IPTV. Esta solução fornece as especificações para uma plataforma fim-a-fim para a implementação de serviços IPTV, e permite a qualquer dispositivo, conforme com a especificação OIPF, aceder a serviços enriquecidos e personalizados seja numa rede controlada pelo fornecedor de serviço ou não. A Figura 2 mostra uma abstração da estrutura da solução, em termo de redes e entidades funcionais na rede residencial. Numa rede residencial podem estar um ou mais *Open IPTV Terminal*

Function (OITF) conetados a um ou mais *gateways*. Estes *gateways* não têm de ser dispositivos separados do OITF podendo todos residir neste, como é o caso numa STB ou TV. A rede de acesso, *access network*, pode ser ou não gerido pelo fornecedor de serviço, mas quando acontece este pode usar mecanismos da rede para melhorar a qualidade dos seus serviços, como a diferenciação do tráfego para priorizar tráfego IPTV, melhorando a qualidade do serviço e a experiência do utilizador, ou *multicast* IP para diminuir a largura de banda usada na distribuição de serviços TV ao vivo. Acessível a partir da rede de acesso, a plataforma de serviço, *Service Platform*, é responsável pela autenticação e gestão de sessão do OITF, e a gestão dos serviços IPTV fornecidos. A plataforma oferece a possibilidade ao OITF de escolher fornecedores de serviços diferentes em função dos serviços pretendidos, sendo a escolha feita usando o *IPTV Service Provider Discovery*, que devolve informação sobre os fornecedores disponíveis.

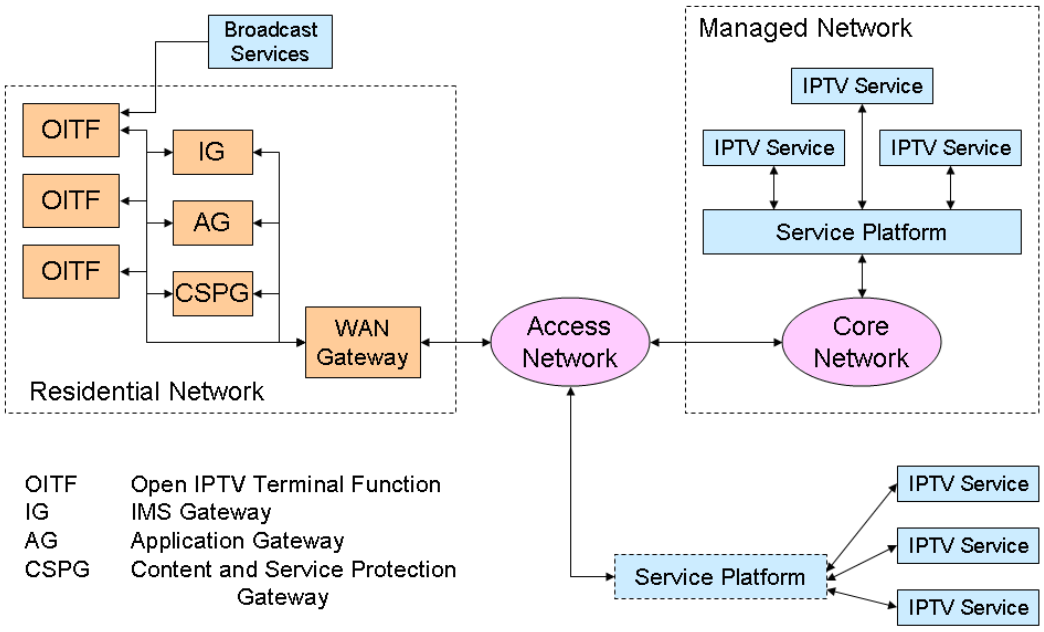


Figura 2: Âmbito da solução OIPF IPTV [11]

A solução suporta os seguintes serviços[11]:

- Televisão ao vivo;
- gravação de televisão em direto ou DVR;
- EPG;
- serviços híbridos, com a combinação de canais de transmissão e IPTV no OITF;
- serviços de VOD;

- serviços de informação, como notícias;
- serviços de comunicação, como notificações e sua mistura com os serviços de entrega de conteúdo.
- serviços *Over the Top (OTT)*, serviços que disponibilizam produtos através da Internet ignorando os métodos de distribuição tradicionais, como serviços de mensagens [1].

Estes serviços são fornecidos através de funcionalidades especificadas que são[11]:

- Fornecimento de serviços, qualquer que seja a rede usada;
- Gestão de utilizadores, incluindo a gestão de vários utilizadores na mesma casa, quando aplicável;
- provisionamento de QoS;
- Gestão remoto dos dispositivos na rede doméstica, incluindo configuração, diagnóstico de falhas e atualização de *software*;
- Serviços de acesso e controlo;
- Navegação de serviço e conteúdo;
- Plataformas para aplicações interativas;
- Proteção de conteúdo e serviço;
- Compatibilidade com dispositivos de rede doméstica que respeitam as normas *Digital Living Network Alliance (DLNA)*

Para a distribuição de média OIPF suporta diversos formatos de áudio e vídeo, como H.264/AVC[13] para vídeo e HE-AAC[13] para áudio , assim como o transporte de conteúdos relacionados como legendas e outros recursos usados em outras partes da solução, sendo de salientar formatos de gráficos e clipes de áudio para as aplicações interativas. São definidos dois formatos para o transporte de conteúdo nomeadamente MPEG-2 *Transport Stream* e o formato de ficheiro *MP4*. Para ficheiros protegidos, são suportados diferentes mecanismos de transporte e proteção.

O dispositivo OITF é responsável pela codificação e descodificação de vídeo e áudio que transmite e recebe, fazendo a gestão do perfil de utilizador e de todos os dados recebidos, recolhidos e transmitidos pelo dispositivo, podendo servir de ambiente de execução para aplicações interativas[11].

1 Segundo a definição adotada na Nota Explicativa que acompanha a Recomendação de 2014 da Comissão Europeia sobre mercados relevantes (nota de rodapé 27, pág. 16) [7]

A solução define dois meios de execução para aplicações interativas, o Declarative Application Environment (DAE) e o Procedural Application Environment (PAE).

O DAE é baseado num *browser web*, usando a especificação *CEA-2014*, também conhecida como *CE-HTML*, e adiciona a este extensões, como propriedades introduzidas em CSS3 e *tags* HTML5 como a *tag video*, oferecendo também suporte a Scalable Vector Graphics (SVG) *Tiny 1.2*. Uma aplicação DAE ou é uma coleção de documentos, *HyperText Markup Language (HTML)* ou SVG, ou um *W3C Widget*, uma aplicação web empacotada para fácil distribuição. Comparativamente a uma pagina *web* tradicional, as páginas *webs* que constituam uma aplicação DAE partilham contexto, estado e recurso, incluindo informações sobre o estado da aplicação como as suas permissões, prioridades e outras informações [12].

As aplicações DAE são obtidas através do serviço de aplicações IPTV, podendo ser feito o descarregamento local destas para execução ou podem ser acedidas como páginas *web*, o serviço de aplicações IPTV funcionando assim como um servidor web.

O PAE é um ambiente de execução para aplicações baseadas em Java que correm no *Application Gateway (AG)*, uma entidade onde aplicações podem ser descarregadas e executadas remotamente, ou no OITF. O PAE é baseado na norma *ETSI TS 102 728*[10][14], da DVB IPTV, designado por *Globally Executable Multimedia Home Platform (GEM)*. As aplicações podem ser centradas no utilizador, como EPG, controlo de gravações ou cliente VOD, aplicações interativas ou serviços de sistema. Este ambiente é essencial para fornecedores de serviços que querem disponibilizar e correr serviços na rede doméstica como a inserção de publicidades no fluxo de conteúdo, a gestão remota dos dispositivos da rede, a medida de audiência, entre outros. Quando executados no *AG*, este ambiente serve principalmente para a gestão dos aparelhos e fluxos de dados na rede mas pode ser usado para a criação de documentos de UI que são transmitidos e executados no PAE de um terminal conectado, servindo de servidor local. Quando instalado no OITF este ambiente é capaz de fornecer aplicações interativas ao utilizador.

As aplicações PAE são obtidas através do serviço de aplicações específicos do fornecedor, sendo feito o *download* local destas, pois estas aplicações correm localmente.

Mais informação pode ser obtido para *OIPF Release 2 v2.3*, na página do OIPF disponível em https://www.oipf.tv/.

2.2.2 *Nagra OpenTV*

A solução *Nagra OpenTV*, designada como *OpenTV Suite*, é uma solução IPTV *end-to-end* que pode ser implementada por qualquer provedor de serviço mas é mais orientado para os atuais provedores de TV, sendo constituído por quatro elementos, o OpenTV OS, usado como *middleware* nas STBs, o OpenTV Platform, para a gestão dos utilizadores e serviços,

o OpenTV Player, para o acesso ao serviço a partir de outros dispositivos, e o OpenTV Experience, a UI da solução .

A solução suporta os seguintes serviços:

- Televisão ao vivo;
- EPG;
- DVR, no dispositivo ou na cloud para acesso remoto (nPVR);
- serviços de VOD;
- *Replay TV* com *startover, timeshifting e catch-up*;
- serviços de notificação;
- aplicações interativas;
- serviços OTT.

O OpenTV *Player* é um leitor multimédia que entrega serviços e conteúdos de maneira seguro, usando DRM, em dispositivos *Android, iOS, Windows e Mac* [25]

O OpenTV *Platform* fornece uma plataforma de gestão de serviços *multiscreen*, para o fornecimento de serviços IPTV a vários assinantes, dispositivos e utilizadores através múltiplas redes [24]. Na Figura 3 é apresentada a arquitetura de uma implementação completa da plataforma. Geralmente, uma implementação usa um subconjunto destes módulos em função das necessidades do fornecedor de serviço.

A plataforma é dividida em três camadas, a camada dos sistemas externos, a camada *NAGRA Head-End* e a camada do utilizador.

A camada dos sistemas externos corresponde aos elementos necessários ao serviço que não são fornecidos pela plataforma, como os fornecedores de conteúdos ou um sistema completo de gestão de assinantes (SMS/CRM). Estes elementos terão de ser completados por sistemas externos a plataforma e integrados nesta pelo fornecedor de serviço, usando a documentação fornecida.

A camada *Nagra Head-End* é dividida em diversos módulos sendo responsável por responder a todo o tráfego gerado pelos sistemas externos e utilizadores. Em função do número de utilizadores, pode ser usado um *head-end* central e outros distribuídos. A comunicação entre o cliente e a plataforma é feito através do *HTTP Router*, responsável pela autenticação do cliente e o *routing* dos seus pedidos, sendo o único elemento com quem este comunica diretamente. Quando a autenticação sucede no *SDP*, o módulo devolve um *token* que o cliente usa para aceder aos outros sistemas. Este módulo serve como *proxy* para aceder a todos os outros módulos da plataforma com quem o cliente pode interagir, sendo de salientar o HDM, CDG, MDS e SDP [25]. Quando diferentes *head-ends* existem, o *HTPP*

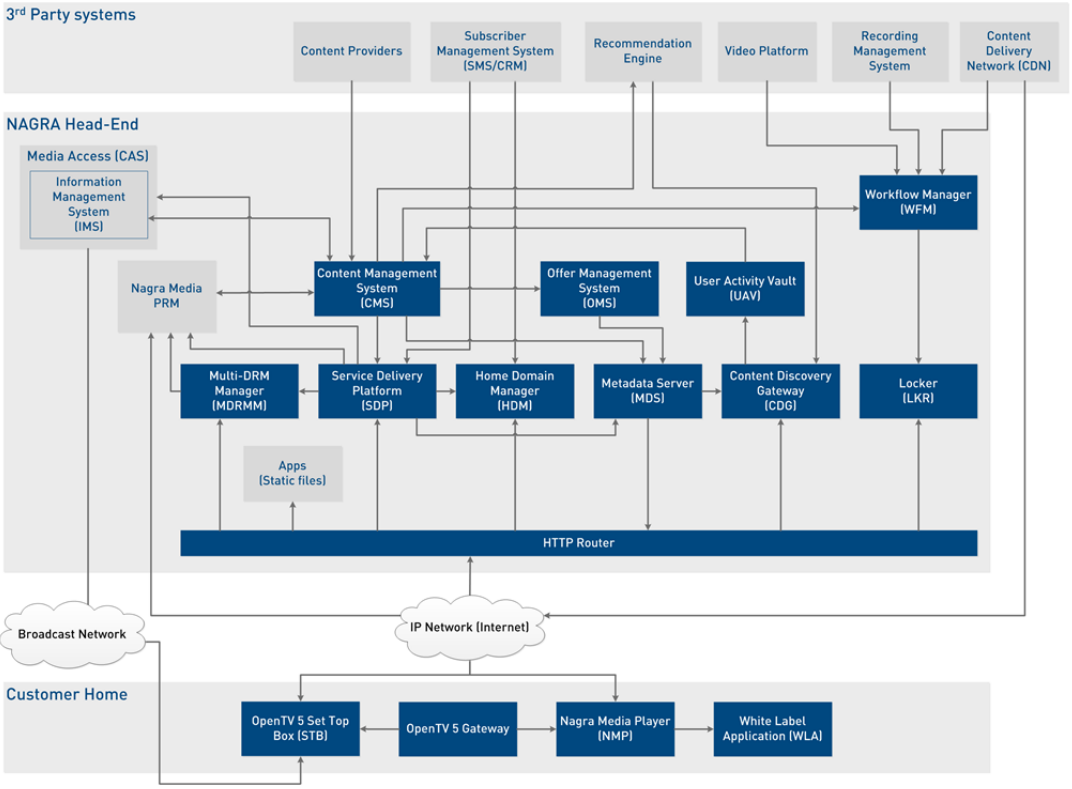


Figura 3: Arquitetura teórica de um sistema para *OpenTV Platform*, segundo *Nagra* .[24]

router é responsável por reencaminhar pedidos para o *head-end* central quando o módulo solicitado não existe localmente. O router também pode ser configurado para bloquear ou restringir alguns serviços, podendo agir como *proxy* reverso. As aplicações criadas para a plataforma são guardados num servidor de aplicações como ficheiros estáticos, sendo depois distribuídas pelos *HTTP router* quando pedidos.

A camada de utilizador define os módulos e dispositivos presentes e que comunicam com o *head-end*. Destes módulos só três podem comunicar diretamente com a plataforma, estes são:

- O OpenTV STB, instalado com o OpenTV OS e disponibiliza serviços como TV ao vivo, VOD e DVR;
- O OpenTV *Gateway*, um *middleware* instalado num *home gateway STB*, que combina as funcionalidades de um router e de uma STB como acesso *WiFi*, serviços VOD e DLNA *Digital Media Server*, permitindo a transmissão de conteúdo a outras STB e conteúdo transcodificado para outros dispositivos;

- O *Nagra Media Player (NMP)*, também designado como OpenTV *Player*, instalado nos dispositivos do utilizador para aceder ao serviço IPTV, podendo comunicar com o *gateway* para aceder ao conteúdo lá guardado.

O módulo *White Label Application* é uma aplicação web que vem com os *middlewares* da plataforma, fornecendo exemplos para a maioria das funcionalidades necessárias para a criação de aplicações interativas, sendo usado como ponto de partida para a criação de novas aplicações para a plataforma. As aplicações são páginas web e usam a *Framework Javascript* fornecido por Nagra para facilitar a integração das aplicações nos elementos da solução, e fornece funcionalidades auxiliares para a UI.

O *OpenTV* OS é uma distribuição Linux arquitectado para portabilidade, modularidade e segurança, sendo construído em torno de uma pilha *Broadcast and Service Information (SI)*, com um sistema de DVR e um módulo de rede doméstico, habilitando a transmissão e recepção adaptável de dados pela internet, através, por exemplo, da tecnologia *Dynamic Adaptive Streaming over HTTP (DASH)*. Os componentes de *middleware* são implementados como serviços Linux e usam o *D-Bus* para comunicação entre processos, o que isola os serviços e permite que módulos de terceiros sejam integrados de maneira rápida e fácil à solução. Os principais benefícios dessa arquitetura permitem o uso da mais recente tecnologia Web, maior robustez e a extensão e modularidade do produto. O *middleware* para a criação de aplicações usa HTML5 para garantir a compatibilidade com uma ampla variedade de aplicações existentes [23].

O OpenTV *Experience* é uma UI que permite aos espectadores pesquisar, selecionar e apreciar uma gama de conteúdo multimédia, tendo sido pensado para ser fácil de usar em qualquer dispositivo [22].

2.2.3 *MediaKind Mediaroom*

MedaKind Mediaroom, é uma plataforma IPTV *end-to-end* proprietária da *MediaKind*, que foi criada pela *Microsoft* e, mais tarde, vendida à *Ericsson* que, depois, criou a *MediaKind*. A solução é implementada com infraestrutura local e entregue através de redes gerenciadas para as STBs dos assinantes, para serviços de TV ao vivo, VOD, *Time Shift* nTS e DVR local ou na cloud. A plataforma suporta os seguintes serviços:

- Televisão ao vivo;
- EPG;
- PiP;
- DVR, localmente ou na cloud;
- serviços de VOD;

- *Instant Channel Change (ICC)*;
- *Time Shift TV*, nTS;
- serviços de notificação;
- serviços OTT.

Alguns destes serviços são disponíveis através de dispositivos como *tablets*, computadores e *smartphones*. Sendo uma plataforma proprietária a arquitetura atual da plataforma não está publicamente disponível mas, pela informação apresentada por *Mariner Partners Inc.*[15], a plataforma é constituída por um cliente *Mediaroom* instalado na STB e um conjunto de servidores que constituem o grosso da plataforma. A arquitetura lá apresentada é equivalente a arquitetura da plataforma *Microsoft Mediaroom* visível na Figura 4.

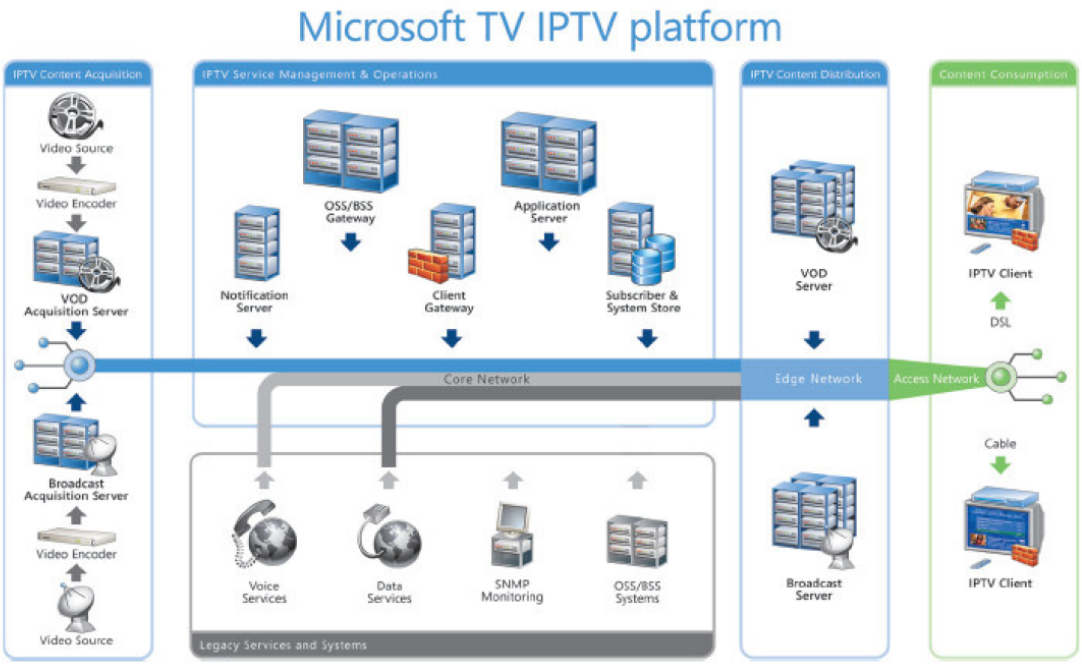


Figura 4: Arquitetura da plataforma Microsoft TV IPTV Edition [29]

Como apresentada na Figura 4, a plataforma é dividida em 4 domínios. No domínio *IPTV Content Acquisition*, os conteúdos VOD e TV ao vivo são codificados e preservados em *Acquisition Servers*. No domínio *IPTV Service Management and Operations*, é efetuado a gestão da plataforma, os seus conteúdos e subscritores. Neste domínio também são guardados, em *Application Servers*, as aplicações interativas. As aplicações são criadas usando o *Application Development Kit (ADK)* da plataforma, baseado na *framework .NET*, e são aplicações *C#* que correm nestes servidores devolvendo ficheiros *markups XML* interpretados nas STBs [18]. As aplicações desenvolvidas para a plataforma são denominadas por Presentation Framework

(PF) *Applications* [3]. No domínio *IPTV Content Distribuition* é feito a distribuição dos conteúdos e serviços da plataforma, sendo usados servidores VOD e de TV ao vivo próximos dos clientes para diminuir a latência. No domínio *Content Consumption* os utilizadores usam uma STB instalada com o cliente Mediaroom para acederem a plataforma. O cliente Mediaroom comunica com a plataforma autenticando-se lá, e serve de *middleware* para as aplicações Presentation Framework (PF).

2.3 DESENVOLVIMENTO DE APLICAÇÕES PARA TELEVISÃO

O desenvolvimento de aplicações para *Smart TVs* segue, geralmente, quatros etapas, aconselhadas pela *Samsung*[8] e especificadas por Arthur Varushyla[4], que são :

- Planificação da aplicação e da sua UI;
- Implementação da aplicação;
- Depuração e teste da aplicação;
- Submissão e publicação.

Todas essas etapas são comuns no desenvolvimento de *software* mas é aqui dado uma grande importância a planificação da UI da aplicação, pois existem requisitos de usabilidades diferentes daqueles presentes nos computadores que devem ser tomados em considerações [5].

2.3.1 *Desenvolvimento de UI para a Samsung*

A *Samsung* aconselha fazer a planificação tendo em considerações quatro pontos[27] que são:

- princípios de *design*;
- a maneira como se interage com a aplicação, geralmente efetuado através do comando (mas pode incluir outros métodos, como o telemóvel);
- a sua aparência em ecrãs de diversos tamanhos e resoluções, uma vez que as televisões variam muito quanto ao seu tamanho e resolução devendo ser assegurada a aparência e usabilidade da aplicação nessas;
- as especificações obrigatórias definidas pela plataforma de distribuição. A televisão é uma plataforma fragmentada [1] sendo que cada distribuidora especifica requisitos que as aplicações devem cumprir.

Os princípios de *design* englobam todos as regras e considerações que devem ser tomadas durante o *design* da aplicação. A *Samsung* divide esses pontos em três categorias que são as características do ambiente de utilização, princípios de *design* da aplicação e regras para fornecer uma melhor experiência ao utilizador.

O primeiro ponto especifica as características próprias ao uso da televisão, um ambiente descontraído, efetuado a uma grande distancia, aproximadamente três metros mas varia enormemente [26], com um controlo reduzido através do comando e podendo ser utilizado por várias pessoas.

O segundo ponto especifica regras, ou boas práticas, que devem ser seguidas pela aplicação e suas interfaces sendo divididas em categorias que são:

- Simplicidade, aconselhando um aspeto simples e prático que facilita a descoberta de conteúdo, limitando o número de transições e navegações, com um uso intuitivo;
- Clareza, a navegação devendo ser simples, sem efeitos complexos e como esperado pelo utilizador, indicando inequivocamente a localização do utilizador na aplicação;
- Controlo pelo Utilizador, a aplicação deve ter em consideração o dispositivo de controlo usado pelo utilizador apresentando uma aparência adequada, para além disso as ações tomadas pela aplicação devem corresponder ao esperado pelo utilizador, não devendo a aplicação impor as suas regras de navegação;
- Consistência, de maneira a facilitar o uso da aplicação esta não se deve desviar das normas pré-estabelecidas na plataforma de uso. Além disso, a aplicação deve manter-se consistente no seu todo de maneira a não confundir o utilizador. A *Samsung* especifica três tipos de consistência que são a consistência de controlo, do *layout* do ecrã e de navegação;
- Resposta, é aconselhado que cada ação efetuada pelo utilizador provoque uma resposta por parte da aplicação, seja com uma mensagem de erro, seja com uma alteração no aspeto visual dessa. Essas respostas não devem confundir o utilizador, devendo ajudá-lo a compreender o que pode ou não fazer. Da mesma maneira, operações demoradas devem providenciar uma animação para confirmar que o ecrã irá mudar como pedido pelo utilizador;
- Considerações estéticas, aplicações para televisões devem ter em atenção as cores usadas, as resoluções e as composições dos ecrãs que são diferentes, por exemplo, de computadores e telemóveis. Televisões têm diferentes resoluções e, mesmo para a mesma resolução, o tamanho e tecnologia do ecrã varia muito. A *Samsung* recomenda criar a aplicação para uma resolução fixa de *1080p*, com o redimensionamento gerido pela *framework*. Para texto recomenda-se um tamanho de fonte mínimo de vinte pixeis, com fontes e cores legíveis em quaisquer ecrãs e a uma boa distância.

No último ponto a *Samsung* especifica todos os requisitos obrigatórios e aconselhados que as aplicações devem possuir para poderem ser distribuídas na sua plataforma. Esses requisitos especificam, por exemplo, a resolução mínima das imagens usadas, o comportamento da aplicação face a *inputs*, como o *player* se deve comportar na aplicação e, também, as respostas que esta deve fornecer ao utilizador.

2.4 SUMÁRIO

Como foi abordado existem diferentes plataformas IPTV e fornecedores de serviços IPTV. Essas plataformas dividem internamente o seu domínio em quatro partes [9] que são os fornecedores de conteúdos, os fornecedores de serviços, a rede de acesso e o utilizador. As plataformas definem como interagem com os domínios e o que fornecem para ter uma solução completa. Mesmo sendo diferentes, todas as plataformas suportam os seguintes serviços:

- televisão ao vivo;
- DVR;
- EPG;
- VOD;
- serviços de notificações.

As aplicações interativas usadas nessas plataformas são criadas usando diferente linguagens, por exemplo, aplicações para a plataforma *Mediaroom* usam *C#* [18] enquanto que é usado uma *framework Javascript* e *HTML5* na plataforma *Nagra OpenTV*[24][23]. O desenvolvimento de aplicações para televisões ou outras soluções IPTV segue quatro etapas que são a planificação, implementação, teste e depuração, e submissão e publicação da aplicação. Dentro de cada etapa é dada muito importância à planificação, mais especificamente a planificação da UI devido ao contexto de utilização. Assim, a *Samsung* aconselha fazer a planificação tendo em conta quatro pontos[27]:

- princípios de *design*;
- as maneiras do utilizador interagir com a aplicação;
- a aparência dessa em ecrãs de diversos tamanhos e resoluções;
- os requisitos mandatórios definidos pela plataforma de distribuição.

3

ANÁLISE E ABORDAGEM DA PLATAFORMA LANDING PAGES

Neste capítulo, é apresentada a abordagem que será usada para a criação de um interpretador da plataforma LP para a nova plataforma da Altice. É inicialmente feito a apresentação da plataforma LP, sendo analisados os grupos de *templates* a recriar e os componentes que os compõe. Depois é apresentada a plataforma para onde o interpretador será desenvolvido e integrado. Finalmente, é apresentada uma proposta para a resolução do problema e uma possível arquitetura para a implementação.

3.1 PLATAFORMA LANDING PAGES

A plataforma LP é uma plataforma para a criação de aplicações interativas para TV baseadas em *templates* e foi criada para reduzir o tempo de desenvolvimento das aplicações, enquadrando-se numa metodologia *no code*, pois esta permite a criação de aplicações vocacionadas a promoção de conteúdos e serviços multimédias, através de *templates* predefinidos, sem ter que se proceder a desenvolvimento adicional.

Desde a sua criação, esta plataforma foi utilizada para criar diversas aplicações, tendo evoluído para integrar mais *templates* e reduziu o tempo de desenvolvimento destas aplicações. Além disso, também permitiu que a equipa de produto de televisão da empresa seja capaz de criar aplicações rapidamente, envolvendo a equipa de desenvolvimento unicamente para a sua disponibilização, dando maior independência a pessoas não especializadas e mais tempo à equipa de desenvolvimento. Esta plataforma, como muitas plataformas *no code*, funciona muito bem para a criação de aplicações enquadradas pelos *templates* especificados e facilita a criação rápida, por pessoas não especializadas, de aplicações interativas para TV, sendo que é previsto a continuação da evolução da plataforma com a introdução de um *template* genérico. A seguir é apresentada a sua arquitetura e uma análise dos seus componentes.

3.1.1 *Arquitetura Plataforma Landing Pages*

A plataforma é composta por quatro componentes principais, apresentados na Figura 5.

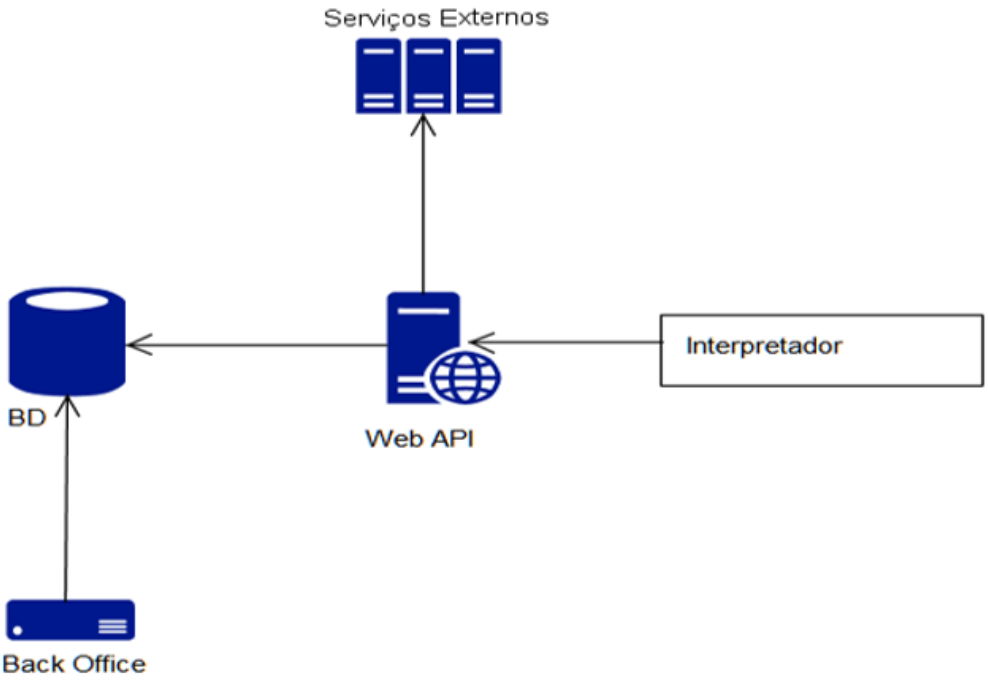


Figura 5: Arquitetura simplificada da plataforma Landing Pages

O *backoffice* permite a criação de uma aplicação interativa, dita aplicação LP, usando um dos cinco grupos de *templates* hoje disponíveis, por exemplo, o *template* do grupo 4, apresentado na Figura 6, pode ser uma aplicação LP válida. Uma vez selecionado o *template*, é possível configurá-lo, escolhendo o *background*, a estrutura dos menus, parte da lógica, as imagens a apresentar, as cores e fontes de letras, entre outras opções. Após configuração, a aplicação é guardada numa base de dados na forma de um conjunto de parâmetros, podendo ser depois modificados.

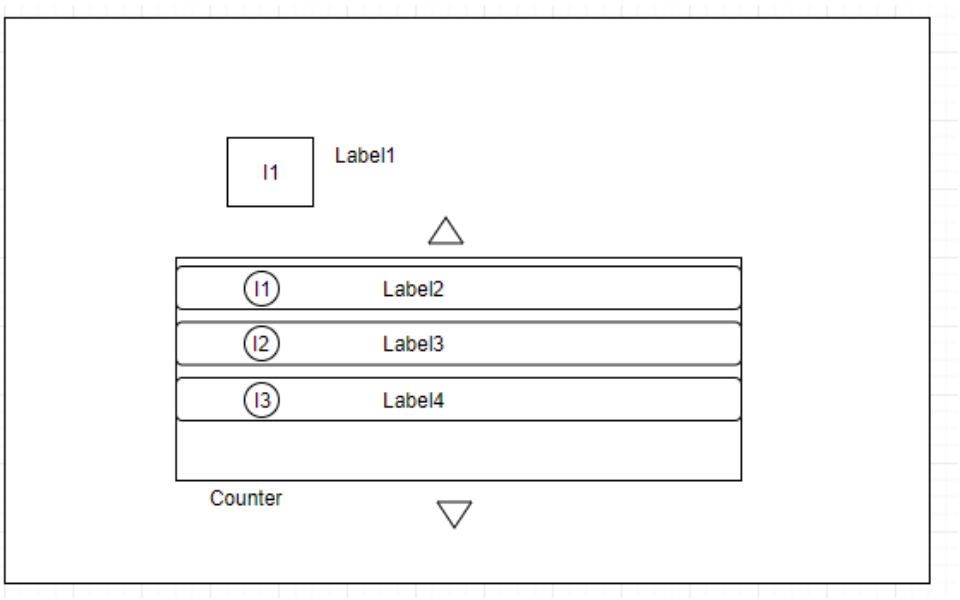


Figura 6: Exemplo de um *template* com um *background* e um menu

As aplicações são depois expostas através de uma *Web API* que recupera da base de dados as configurações, usando também serviços externos caso necessário. O resultado de um pedido à API pode ser *JavaScript Object Notation (JSON)* ou *eXtensible Markup Language (XML)*. O interpretador é o adaptador para a plataforma onde as aplicações devem correr. Ele acede à API, recuperando a informação sobre a aplicação e interpreta-a, criando a aplicação configurada. É uma parte essencial da plataforma e é uma grande fonte de flexibilidade, pois permite o uso das aplicações em qualquer plataforma que implemente o interpretador.

3.1.2 *Aplicação Landing Pages*

Como definido na secção 1.4, uma aplicação LP é uma aplicação interativa construída usando um conjunto de *templates* relacionados, usados em conjunto para a apresentação e o acesso a multimédia. Na plataforma, uma aplicação LP é guardada como um conjunto de ficheiros de configurações relacionados, como visível na Figura 7.

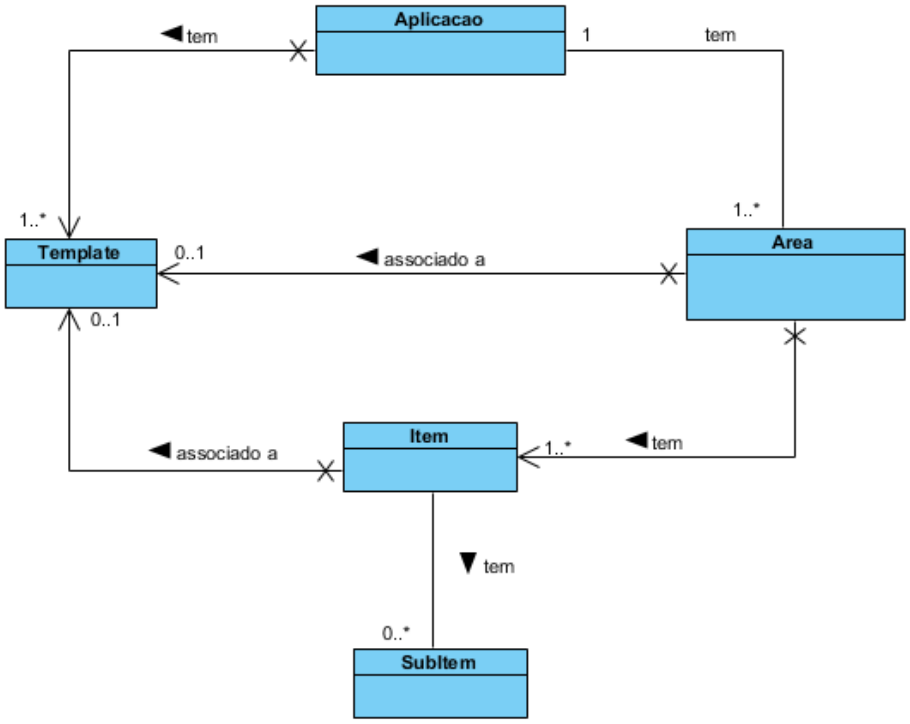


Figura 7: Árvore de configuração de uma app

Pela Figura 7, uma aplicação é constituída por pelo menos um *template*, que é a sua *home page*, várias áreas, que correspondem a subpáginas da aplicação e a quem é associado um *template*, itens, que são conteúdos associados a uma área e que podem ser associados a *templates* da aplicação, e subitens que são conteúdos dos itens. Áreas e itens possuem um tipo, que especifica o comportamento e a origem dos dados que representam, por exemplo, se uma área é associada a uma *feed* Rich Site Summary (RSS), então os seus itens são os elementos da *feed*. Áreas, itens e subitens possuem entradas diferentes na API, o tipo influenciando também a entrada da API usada para os recuperar.

O primeiro ficheiro de configuração recuperado é a configuração geral da aplicação, tendo informações gerais sobre esta, como os *templates* que usa, a sua identificação, restrições de acesso, entre outras informações. A partir desta configuração todos os *templates* da aplicação são especificados. A primeira página da aplicação é a sua *home page* e corresponde ao *home template* especificado. A partir da *home page* o utilizador navega na aplicação, acedendo as páginas e conteúdos dessa.

3.2 GRUPOS DE TEMPLATES DA PLATAFORMA LANDING PAGES

Os *templates* da plataforma LP são páginas compostas por componentes e especificam como esses são organizados e se comportam . Para facilitar a configuração, os *templates* são otimizados para uma resolução de 1280x720 pixeis, sendo deixado ao interpretador o redimensionamento destes. Todos os *templates* permitem a modificação do *background*. A seguir são apresentados e explicados os grupos de *templates* listados nos objetivos da secção 1.4, sendo apresentado um resumo dos componentes usados nos *templates*.

3.2.1 *Grupo 1*

O grupo 1 só possui um *template*, apresentado na Figura 8.

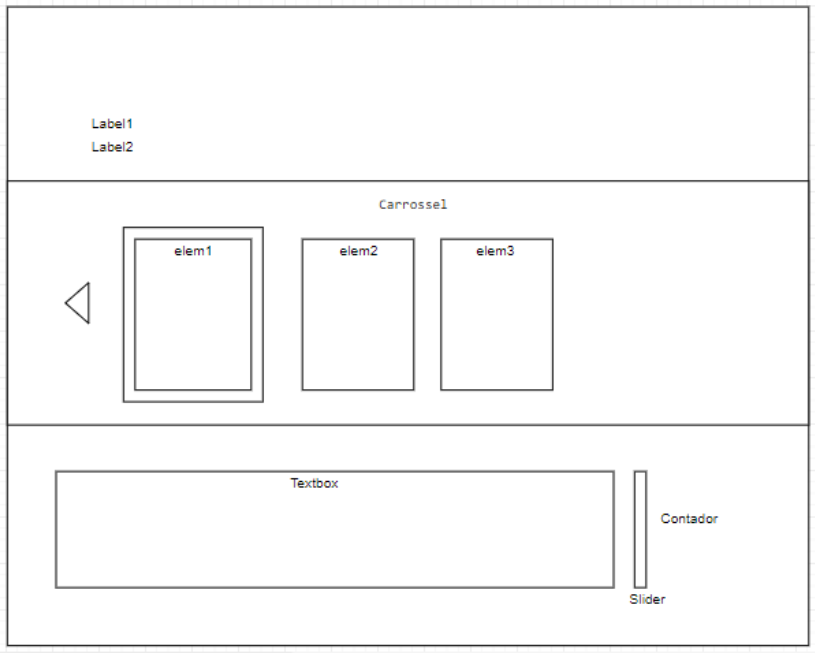


Figura 8: Template do grupo 1

Este *template* é constituído por uma seta, um carrossel, dois *labels*, uma borda, uma *textbox*, um contador e um *slider*. O carrossel é usado para a navegação numa lista de elementos sendo, por omissão, não circular. A posição no carrossel é assinalada pelo contador. A seta assinala o início do carrossel e que se o utilizador continuar a navegar para a esquerda, esse sairá da aplicação. Uma borda é associada ao elemento selecionado, sendo informações sobre esse apresentado na *textbox*. Os *labels* são usados para apresentar informações sobre o elemento como o seu nome. O *slider* está associado a *textbox*, representando a navegação nele caso necessário. O carrossel usa como fonte de dados os itens da área associada. A

posição e tamanho dos componentes podem ser modificados, sendo que, para além disso, é possível modificar a fonte e o tamanho de letra a usar para os *labels*, o contador e a *textbox* e a aparência do *slider* é modificável. Para os elementos do carrossel é possível configurar o tamanho desses e o espaçamento entre elementos. A aparência do elemento é a imagem de *background* definido no item que o representa.

3.2.2 *Grupo 2*

O grupo 2 possui um único *template*, apresentado na Figura 9, sendo este a *home page* das aplicações deste grupo. Essas aplicações usam *templates* do grupo 6 para complementar a aplicação.

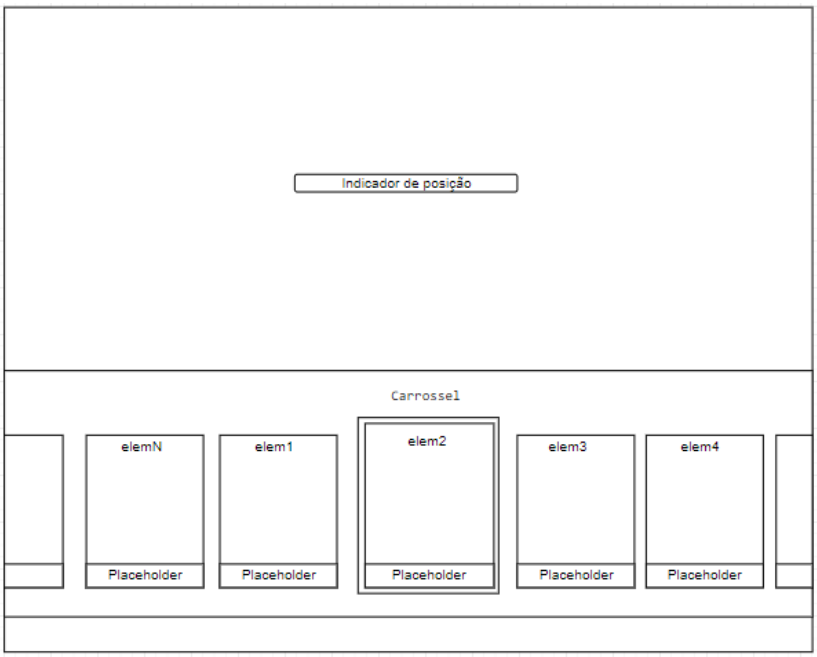


Figura 9: Template do grupo 2

O *template* possui três componentes, um carrossel, que por omissão é circular, um indicador de posição e uma borda. O carrossel é preenchido usando as áreas associadas à aplicação. Os elementos do carrossel possuem um *placeholder* para o nome ou descrição do elemento. O elemento selecionado é assinalado com uma borda e é ampliado, sendo a posição do elemento no carrossel assinalada pelo indicador de posição. A posição, tamanho e aparência do indicador de posição são configuráveis. Para o carrossel é possível modificar a sua posição, se é ou não circular e o tamanho dos seus elementos. Além disso, o espaçamento entre elementos, a posição do *placeholder* e o estilo do texto, ou seja fonte, tamanho de letra e cor, presente nesse, são configuráveis.

3.2.3 *Grupo 3*

Este grupo possui dois *templates* mas o segundo é um caso particular do primeiro, sendo usados juntos e considerados como um único *template* na plataforma e são, assim, a *home page* das aplicações desse grupo. O primeiro *template*, apresentado na Figura 10, possui um carrossel, duas setas, quatro *labels* e um contador.

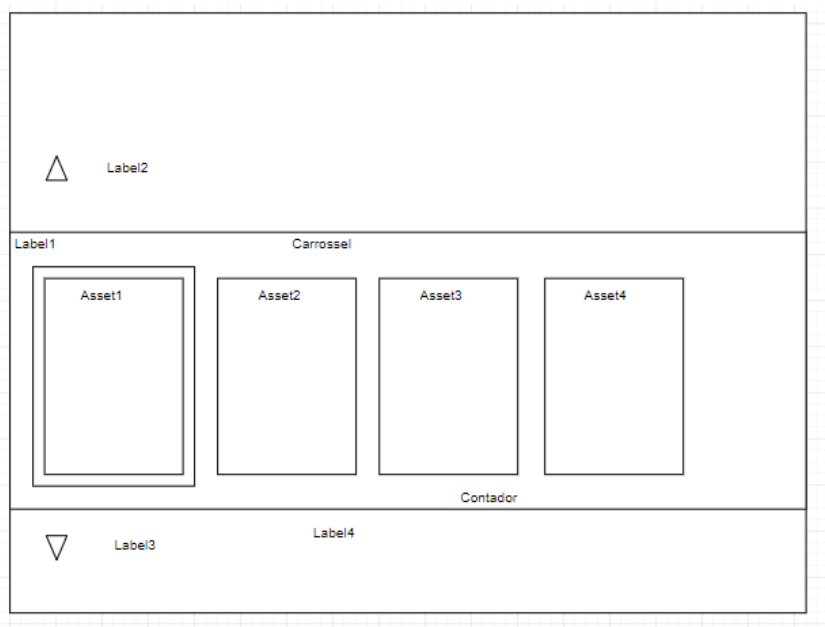


Figura 10: Primeiro template do grupo 3

As setas e os *labels* ao lado delas indicam o nome do próximo ou anterior item que será apresentado se o utilizador navegar para lá, sendo semelhante a um menu vertical. O *label* 1 é o nome da área agora apresentada sendo os itens da área usados para preencher o carrossel. O *label* 4 é o nome do elemento selecionado. O carrossel pode ou não ser circular, não o sendo por omissão. Como no *template* 1 do grupo 1, o elemento selecionado é assinalado por uma borda configurável, e o contador especifica em que elemento do carrossel o utilizador se situa. Quando a área possui um único item é usado o segundo *template*, apresentado na Figura 11, ficando o carrossel com um único elemento e sendo adicionados dois *labels* e uma *textbox*.

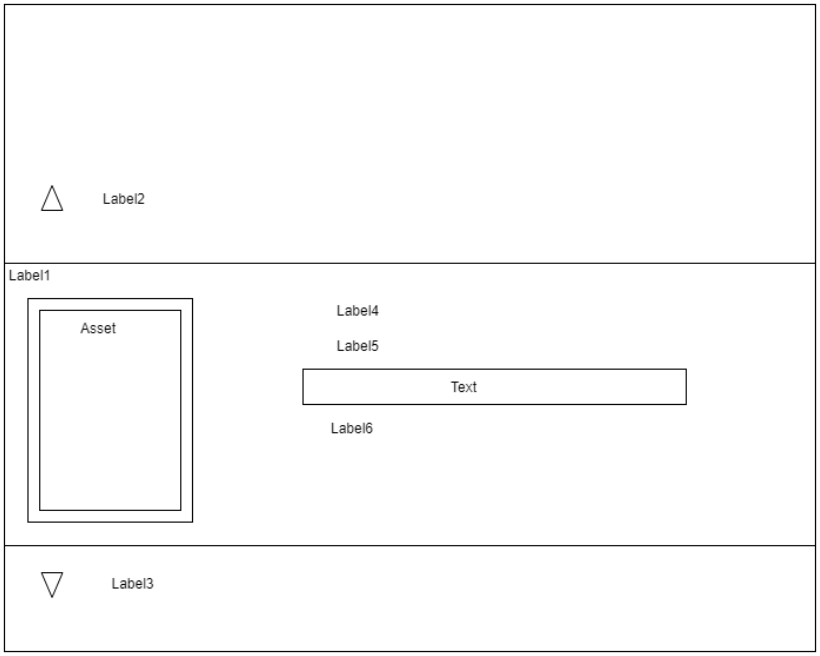


Figura 11: Segundo template, variação do primeiro template do grupo 3

Esses *labels* dão mais informações sob o elemento, sendo que a *textbox* é usada para dar uma descrição mais extensa desse. O contador presente no primeiro *template* é escondido neste *template*, passando a ser desnecessário. Em ambos os *templates* é possível alterar o tamanho e posição do contador, carrossel e os seus elementos, da *textbox*, das setas e dos *labels*. Nos *labels* e na *textbox* é possível alterar a fonte, tamanho e cor de letra usado.

3.2.4 *Grupo 4*

Neste grupo existem dois *templates*. O primeiro *template*, apresentado na Figura 12, possui um menu vertical, um ícone, um *label*, um contador e duas seta, e é a *home page* das aplicações deste grupo. O menu contem elementos que possuem um *label* e um ícone associado, sendo preenchido com as áreas associadas a aplicação. As setas e o contador são relacionados com o menu, variando quando o utilizador navegar nele, por exemplo, a seta por cima do menu desaparece quando o utilizador estiver no primeiro item do menu. O ícone e *label* dão informações sobre a aplicação e página.

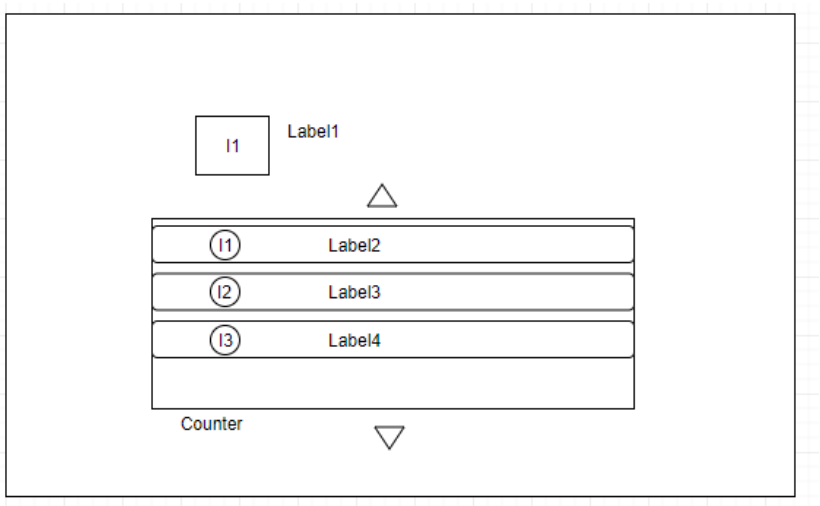


Figura 12: Primeiro template do grupo 4

Neste *template* é possível configurar o tamanho e a posição de todos os componentes, para além disso, o formato do contador e o estilo desse e das *labels* são modificáveis. A imagem que representa as setas é configurável. O tamanho e o espaçamento entre elementos do menu são configuráveis.

Como é visível na Figura 13, o segundo *template* possui um carrossel, um contador, um *label*, um ícone, uma seta e uma borda. A seta, a borda e o contador são associados ao carrossel, sendo que a borda sinaliza o elemento do carrossel selecionado, o contador indica em que elemento do carrossel o utilizador se situa, e a seta sinaliza que se continua a navegar para a esquerda quando chegar ao primeiro elemento, irá sair da página. Os elementos do carrossel possuem um *asset* e um *placeholder* associado. O *asset* é geralmente uma imagem.

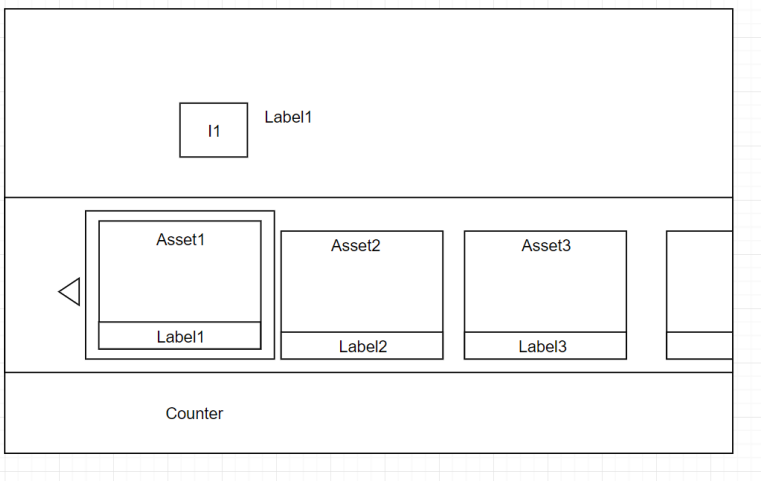


Figura 13: Segundo template do grupo 4

É possível configurar o tamanho e a posição de todos os componentes da página e os elementos do carrossel, para além disso, o formato do contador e o estilo desse e dos *labels* são modificáveis, podendo ser alterado a imagem que representa a seta. O tamanho dos elementos do carrossel e o espaçamento entre eles são modificáveis.

3.2.5 *Grupo 5*

Este grupo é constituído por dois *templates*, o primeiro *template*, apresentado na Figura 14, e o segundo *template*, apresentado na Figura 15.

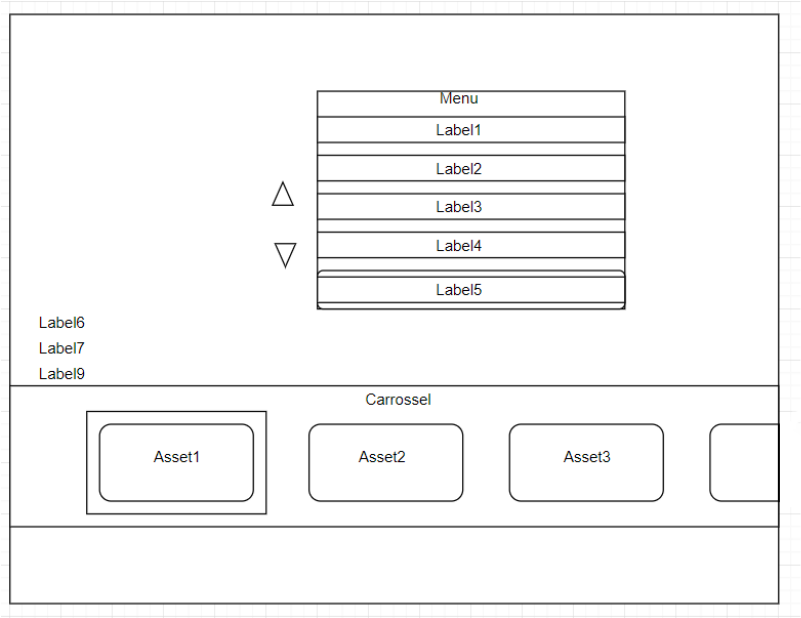


Figura 14: Primeiro template do grupo 5

O primeiro *template* é constituído por um menu vertical, duas setas associadas ao menu, três *labels*, um carrossel e uma borda que identifica o elemento do carrossel selecionado. O menu é preenchido usando as áreas associadas a aplicação, e o carrossel com os itens associados ao item do menu selecionado. Os *labels* dão mais informação sobre o elemento do carrossel selecionado. Este *template* é a *home page* das aplicações deste grupo.

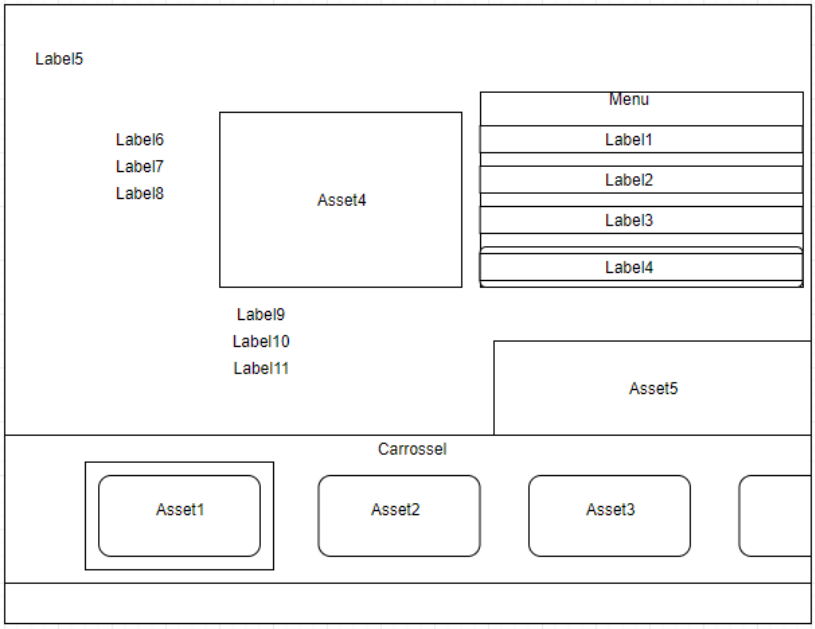


Figura 15: Segundo Template do grupo 5

O segundo *template* é constituído pelos mesmos componentes do que o *template* anterior, sendo adicionado a este duas imagens, *assets*, e quatro *labels*. O carrossel é constituído pelos itens associados ao elemento selecionado na página anterior, sendo igual ao carrossel do primeiro *template*. O menu é preenchido com os itens associados ao elemento do carrossel selecionado e o *asset 4* é a imagem desse elemento, sendo o *asset 5* uma imagem associado a página. Os *labels* dão informação sobre o conteúdo escolhido. Neste grupo os carrosséis não são circulares. Como nos *templates* dos grupos anteriores é possível alterar o tamanho e a posição dos diferentes componentes. Para os menus e carrosséis é configurável o tamanho dos seus itens, as suas posições, o espaçamento entre eles e o estilo deles, como o tamanho, a fonte e cor de letra usado, sendo este último ponto também configurável nos *labels*.

3.2.6 *Grupo 6*

Este grupo possui dois *templates*, o *template* 1, Figura 16, e o *template* 2, Figura 17, mas nenhum deles pode ser usado como *home page*, sendo usados por aplicações dos outros grupos.

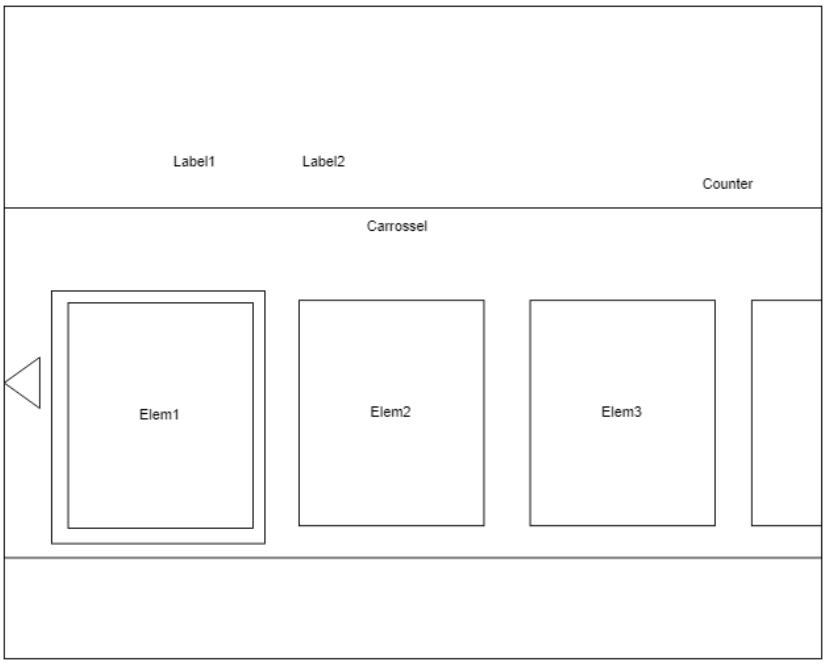


Figura 16: Primeiro template do grupo 6

O primeiro *template* possui um carrossel, dois *labels*, um contador, uma seta e uma borda para sinalizar o elemento selecionado. O carrossel é preenchido com itens relacionados com a página e o item que a criou. Os *labels* dão informações sob a página e o elemento selecionado. O contador indica em que elemento do carrossel o utilizador está. A seta sinaliza ao utilizador que se este continuar a navegar para a esquerda ele irá voltar para a página anterior. O carrossel por omissão não é circular.

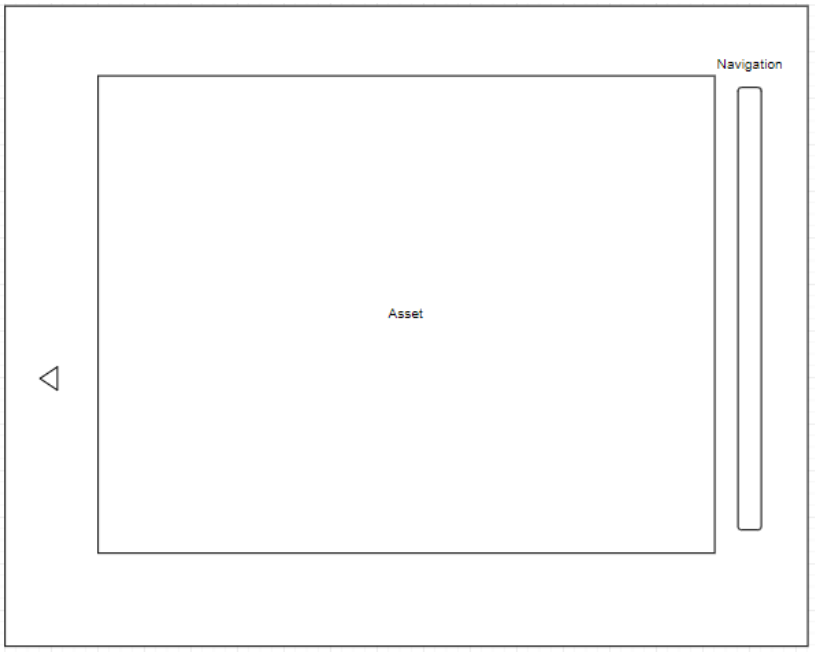


Figura 17: Segundo template do grupo 6

O segundo *template* possui uma seta, uma barra de navegação e uma imagem (*asset*). Este *template* recria um *slideshow*, permitindo ao utilizador navegar entre as diversas imagens apresentadas. A seta sinaliza ao utilizador que navegar para a esquerda, a partir do primeiro elemento, o irá fazer sair da página, regressando a página anterior.

O tamanho e a posição dos componentes dos *templates* podem ser alterados. Para os *labels* é também possível alterar o tamanho, a fonte e a cor do texto usado. É possível definir o formato do contador, assim como a fonte, o tamanho e a cor do texto usado para este. A barra de navegação é configurável de maneira a modificar a sua aparência.

3.2.7 *Síntese*

Como apresentado anteriormente, existem dez *templates* na plataforma, divididos em seis grupos. Esses usam um número relativamente reduzido de componentes, sendo a maior diferença entre eles a lógica interna própria a cada grupo e *template*. Na Tabela 1 são apresentados os elementos usados e a lista das configurações aplicadas a estes.

Nome	Configurações
Label	posição tamanho, cor e fontes de letra
Carrossel (sem placeholder)	posição tamanho dos elementos espaço entre elementos se é ou não circular
Carrossel (com placeholder)	posição tamanho dos elementos espaço entre elementos se é ou não circular fonte, cor e tamanho do *placeholder* posição do *placeholder* *background* do *placeholder*
Menu Vertical	posição número de itens visíveis tamanho dos itens espaço entre itens *background* do item
Seta	posição tamanho imagem
Contador	posição tamanho, cor e fontes de letra
Borda	posição tamanho aspecto
Icon	tamanho posição imagem
Textbox	tamanho posição
Slider	posição tamanho aparência
Indicador de navegação	posição tamanho aparência
Asset	posição tamanho imagem

Tabela 1: Lista dos elementos e das suas configurações

3.3 PLATAFORMA DE DESENVOLVIMENTO

A nova plataforma da Altice é uma aplicação maioritariamente escrita em JavaScript que usa as capacidades do *browser*, como a API *History*[20], para a criação duma UI responsiva e para fornecer serviços IPTV. Novas funcionalidades são adicionadas à plataforma através de módulos escritos em JavaScript que são registados e descarregados localmente antes de serem carregados, como apresentado na Figura 18 . A gestão dos módulos é feita, no momento, por um ficheiro de configuração e a classe *ModuleManager*. A cada módulo é associado um caminho único na plataforma, permitindo navegar para eles.

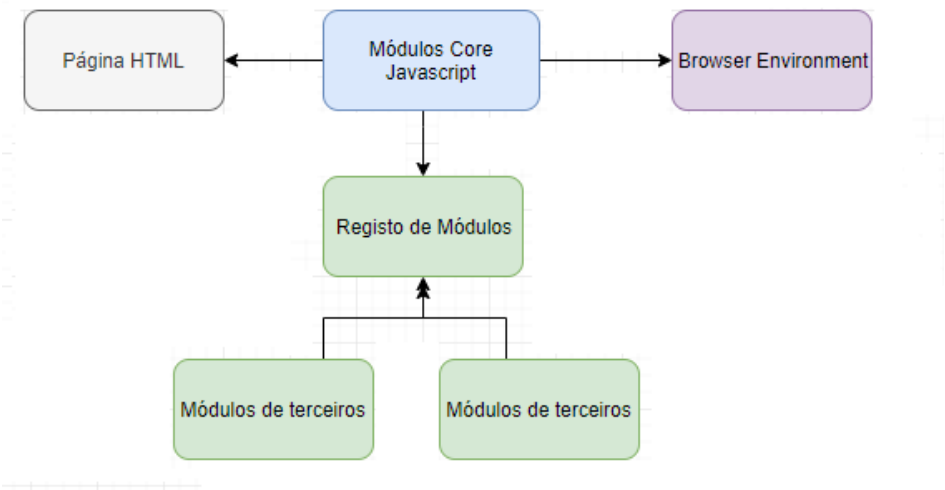


Figura 18: Os módulos são baixados e carregados localmente

3.3.1 *Criação de* User Interface

Uma única página *HTML* é usada pela plataforma, sendo a UI criada por manipulação do *Document Object Model (DOM)* e é constituída por *screens* e *views*.

Uma *view* é um objeto constituído por outras *views* e elementos primitivos da plataforma. Os elementos primitivos são classes da plataforma que abstraem funcionalidades *HTML* como a apresentação de texto e imagens. Uma *view* e os seus filhos são depois traduzidos numa árvore DOM de elementos *HTML*, sendo usado a *tag div* para representar a *view* e os seus filhos. Cada *view* é responsável pelos seus filhos, definindo o estilo desses como o tamanho, a posição e visibilidade, tendo a capacidade de modificá-los.

Uma *screen* é um objeto constituído por uma coleção de *views* e define a UI da página onde o utilizador se encontra. Esta é responsável por responder aos inputs do utilizador, propagando esses aos seus filhos, se necessário. A *screen* define os seus filhos, o estilo desses e passa para esses dados e eventos. Quando uma *screen* é carregada essa é traduzida

num elemento *div*, sendo-lhe acrescentado os seus filhos. Algumas *views* definidas pela plataforma não são acrescentados como filhos, sendo adicionados diretamente a página, como, por exemplo, o *video player*. Não podem ser usados elementos primitivos como filhos de uma *screen*.

As *views* e *screens* são objetos com estados, o estado deles variando em função das interações com o utilizador. Uma *screen* possui dois estados, visível e escondido. Quando visível a *screen* tem o foco, sendo os eventos direcionados para ele, e os seus filhos passam a ser visíveis. Quando escondido a *screen* e os seus filhos são escondidos, não recebendo eventos. As *views* possuem os mesmos estados mas é possível definir novos estados para esses, como o estado selecionado, e transições entre estados, algo não possível para as *screens*.

Todos os componentes são posicionados de maneira absoluta, isto é, a propriedade *CSS positioning* é posta a absoluto e são posicionados de acordo com os valores em pixeis atribuídos as propriedades *top* e *left*. De maneira a não bloquear o fluxo de vídeo e as outras operações da STB, todas as operações bloqueantes ou que podem vir a ser canceladas, como as animações, devem ser assíncronas, sendo usado a API *Promise* [21] e ferramentas fornecidas pela plataforma.

3.3.2 *Navegação entre* screens

A navegação na plataforma é feita através de um *router* que usa a API *History*[20] para recriar a navegação do browser sem o refrescar, guardando a última *screen* apresentada. Para navegar para outra *screen*, uma *screen* usa a API do *router*, passando-lhe o nome do módulo e dados. Esta navegação é feita entre *screens* sendo de dois tipos, entre módulos ou dentro do módulo.

Quando a navegação é feita entre módulos, o *router* geral da plataforma determina se o caminho pedido existe e, caso existir, muda a localização do *browser* e entrega os dados passados ao *router* do módulo que irá determinar que *screen* quer carregar.

Quando a navegação é feita dentro do módulo a localização do *browser* não é alterada, ficando a cargo do router do módulo determinar a *screen* a apresentar.

Em ambos os casos, se a *screen* escolhida já tinha sido carregada então é atualizada e apresentada pelo *router*, senão a nova *screen* é carregada e a *screen* até agora visível passa a ficar escondida.

3.4 Criação do interpretador

Como visto anteriormente, a UI da plataforma é representada por *screens* carregadas através de um *router*. Pela estrutura duma aplicação LP e da plataforma, é possível adotar o modelo *Model View Controller (MVC)* para o interpretador, onde o modelo será representado pelas

áreas, itens e subitens da aplicação, as vistas pelos *templates* e as correspondentes *screens*, e o controlador geral constituído, no mínimo, pelo *router* do módulo. Assim, para cada *template* será criada uma *screen*, sendo implementados os componentes dos *templates*, como imagens e menus, através de *views*, reutilizando-se os componentes já definidos na plataforma.
A plataforma permite o registo de várias rotas para cada módulo, podendo cada uma delas corresponder a um dos *templates* ou, pode-se registar uma única rota onde se decida que *template* deve ser apresentado, em todos os casos, o controlador, a partir de informações que recebe e recolhe da API, irá decidir que *template* e com que configuração deve ser carregado. As *screens* recebem eventos da plataforma respondendo a esses alterando as suas *views*, assim essas também poderiam ser implementadas seguindo o modelo MVC onde seriam os controladores, recuperando e processando informações da API que passariam depois as suas *views* para apresentação.

O controlador geral é o único componente que comunica com a *web API*, devendo tentar minimizar os pedidos efetuados.

3.5 SUMÁRIO

Como foi apresentado, a plataforma LP é dividida em diversos componentes sendo de salientar um *backoffice* para a criação de aplicações, uma API para obter as aplicações e o interpretador que reproduz as aplicações no destino. As aplicações são constituídas por *templates* divididos em seis grupos sendo que cada *template* tem as suas propriedades e componentes. Na *framework* a UI é constituída de elementos designados por *views* que, em conjuntos, constituam uma *screen*. Para a criação do interpretador é assim proposto seguir um modelo MVC onde cada *template* será implementado como *screen*, ligados através de um controlador para recriar as aplicações.

4

DESENVOLVIMENTO

Após a análise dos *templates* efetuado na secção 3.2 uma parte do trabalho a ser efetuado já tinha sido identificado. A partir daí foi necessário compreender como um *template* seria implementado na plataforma, compreendendo como interage com a API e os outros *templates*. Para tal criou-se uma prova de conceito, um esboço inicial, onde é implementado um único *template* que recupera e trata dados da API. Este é muito simples consistindo duma única *screen* com diversas *views*, que realiza todo o trabalho, comunicando com a API e guardando todas as configurações e dados recuperados. O esboço só foi criado para melhor compreender a plataforma de desenvolvimento e as interações entre as aplicações e a API mas permitiu, para além disto, identificar diferentes problemas e ajudou a desenhar uma arquitetura básica para o módulo.
Neste capítulo é detalhado como o desenvolvimento do módulo foi efetuado, descrevendo a arquitetura adotado para este, os elementos desenvolvidos, os problemas encontrados e as decisões tomadas.

4.1 PROBLEMAS E DECISÕES

Nesta secção são apresentadas dificuldades e problemas identificados durante a criação do protótipo inicial e do módulo, assim como as respetivas soluções adotadas durante o desenvolvimento para as ultrapassar. A maioria desses problemas devem-se a uma falta de conhecimento e experiência sobre a plataforma. Os problemas identificados mais impactantes foram:

- Um motor de animação reduzido, limitado a propriedades específicas que dependem do elemento que se quer animar;
- Um motor de escala inexistente, não permitindo um ajuste automático dos elementos visuais;
- A manutenção dos elementos visuais criados, estes permanecendo na página até ao seu próximo uso o que dificulta a configuração desses;

- As mudanças na plataforma devido a atualizações;
- A falta de uma documentação extensa que explique os diversos elementos da plataforma de desenvolvimento;
- A navegação entre *screens* tem as suas dificuldades.

Esses problemas são a seguir discutidos com mais detalhe e apresentados as soluções adotadas.

4.1.1 *Motor de animação*

As animações são uma parte essencial de uma interface interativa, permitindo mostrar ao utilizador que algo mudou devido às suas ações, sendo muito utilizado nos diversos grupos de *templates*. A plataforma permite a criação de animações de duas maneiras:

- estaticamente, com a definição de estados e transições nos elementos com valores fixos;
- dinamicamente, com a criação e execução em *runtime* de animações.

Estas duas maneiras complementam-se mas são prejudicadas pela falta de maturidade do motor, pois este só permite a animação de um número reduzido de propriedades que dependem muito do elemento que se quer animar. Além disto, só é possível animar uma propriedade de cada vez aumentando o número de animações a criar para a realização de animações complexas. Estas também não implementam eventos, impedindo a realização de ações no fim das animações. Para resolver esses problemas foi necessário substituir as animações indisponíveis por outras animações, por exemplo, em vez de animar a passagem de uma cor para outra num elemento, são criados dois elementos, um com a cor original e outro com a cor pretendida, e é feito a transição de um para outro através da opacidade desses. Estas substituições não são realizadas para todas as propriedades que se quer animar. Como a plataforma não implementa um mecanismo de execução no fim da animação de uma ação, foi decidido criar uma classe que estende as animações criadas pela plataforma para implementar tal mecanismo. Esta classe adiciona uma propriedade que representa uma função, geralmente chamada por *callback*, que é executada, se existir, sempre que a animação acabar, sendo-lhe passado como argumento a própria animação.

4.1.2 *Motor de Escala*

A plataforma foi desenvolvida para a apresentação de conteúdo HD de uma resolução de *1080p*, não disponibilizando, neste momento, de mecanismos para o redimensionamento automática das UIs criadas. Como os *templates* são especificados para um resolução de *720p*

é necessário fazer o *scaling* destes, não para *1080p* mas para a resolução do ecrã onde são apresentados. Para isto, foi decidido calcular o rácio entre a resolução de referência e a do ecrã, sendo depois aplicado um factor de escala (*scale*), através da propriedade *transform*, diretamente à UI usando o rácio calculado. Para ter um dimensionamento uniforme o ratio é calculado relativamente a largura do ecrã.

4.1.3 *Gestão dos elementos visuais*

Os elementos visuais, como as *screens* e as *views*, são geridos pela plataforma, sendo preservados depois de serem escondidos. Este comportamento leva a que as *screens* só são criadas uma vez, preservando o seu estado mesmo depois deste ser escondido. Se isto permite evitar a criação e disposição sucessiva de objetos também leva a um problema quando a mesma *screen* é utilizada por diversas aplicações com configurações distintas, pois quando a *screen* volta a ser apresentada, esta é apresentada com uma configuração antiga, até que a nova configuração e os novos dados chegam. Para contornar este problema é usado um *overlay* que esconde a *screen* até à nova configuração ser aplicada, sendo as alterações forçadas onde necessário. Além disto, os elementos mais dinâmicos das *screens*, como os carrosséis e menus, são limpos quando escondidos. Se tal permite evitar dados desatualizados na página, também leva a que esses elementos podem voltar a ser preenchidos com os elementos limpos, podendo provocar a criação desnecessário de objetos.

4.1.4 *Alterações devido a atualizações*

A plataforma ainda se encontra em desenvolvimento não possuindo uma versão estável, sofrendo periodicamente alterações que modificam a maneira como os elementos são apresentados e utilizados, para além de remover alguns componentes. Estas atualizações, se esperadas, causam muitas vezes problemas na UI. Tais alterações são inevitáveis, devendo ser tratadas à medida que a plataforma é atualizada. Mesmo assim é possível limitar os problemas causados e a extensão das modificações a efetuar usando abstrações e dividindo corretamente as funcionalidades e os elementos visuais.

4.1.5 *Falta de documentação*

Por ser uma plataforma proprietária em desenvolvimento a documentação, se existente, é limitada, não explicando os diversos componentes da plataforma, como interagem e como podem ou devem ser usados. Tal falta foi muito sentido ao longo da prototipagem, o que levou a criação de uma *mini* documentação com apontamentos sobre os componentes mais importantes, os seus métodos, as suas variáveis, como eles se comportam, o que

conseguam ou não fazer e como podem ser usados. Tais apontamentos foram essenciais na implementação, durante a qual foram complementados, mas não são suficientes para compreender toda a plataforma, não substituindo uma documentação oficial.

4.1.6 *Navegação entre screens*

Devido a falta de documentação foi difícil perceber como se podia passar de uma *screen* para outra preservando, no histórico, a *screen* anterior. Para além disto, as aplicações LP navegam usando dados provenientes da API, devendo possivelmente passar informação a *screen* que lhe segue e preservar o estado de alguns dos seus elementos visuais, sendo essa última informação usada quando o utilizador voltar para a *screen*. Sem documentação a mão decidiu-se inspecionar o código para encontrar as classes e métodos a usar, verificar o que as variáveis definidas representam e modificá-las no módulo para obter o comportamento esperado. Depois de compreender como o histórico era usado e como a navegação é feita internamente foi necessário definir como o *router* iria escolher a *screen* a mostrar. Como um *router* pode possuir várias rotas seria possível definir uma rota para cada *template* existente mas isto implica a criação e registo de novas rotas, para além da lógica necessária para aceder a esta, cada vez que um novo *template* é criado. Decidiu-se, por isso, criar uma única rota no *router* deixando a escolha ao controlador através das informações recebidas.

4.1.7 *Sumário*

Como visto, muito dos problemas derivaram da plataforma não estar estável e só foram superados através de um estudo contínuo do código fonte, a criação de uma documentação própria e a organização do módulo de maneira a evitar o uso direto dos elementos da plataforma. Outros problemas sugiram relativamente à organização dos componentes, relativamente à lógica a implementar e a representação dos dados, que serão explorados na próxima secção.

4.2 IMPLEMENTAÇÃO

A implementação começou com a criação dum esboço onde foi implementado um único *template* para familiarizar-se com a plataforma. Para tal, foi criado um *router* que regista uma única *route* que devolve sempre a mesma *screen*, implementando o mínimo necessário para o bom funcionamento do *template*. Este primeiro esboço permitiu identificar os métodos necessários para recuperar dados da API, como podem ser recuperados pela *screen* assim como podem ser localmente conservados para evitar pedidos desnecessários. A partir deste, a implementação do módulo foi feito iterativamente e incrementalmente, começando pela

criação de um controlador que faz a ligação entre o *router* e as *screens* do módulo. A lógica que era implementada na *screen* passou a ser feita no controlador, sendo que a *screen* interage com este para recuperar dados. O controlador foi criado para poder manter o contexto da aplicação entre *screens* e abstrair a lógica de recuperação dos dados das mesmas.
Com a criação do controlador e a implementação do esboço foram identificados diferentes elementos e partes lógicas que constituem o módulo, sendo esses:

- A lógica de comunicação com a API através do controlador e classes auxiliares;
- A lógica de navegação do lado do *router*, para escolher a *screen* a apresentar e recuperar os dados que lhe serão necessários através do controlador;
- Os elementos visuais, como as *screens* e as *views*, e a lógica própria a cada um deles;
- A lógica de navegação do lado das *screens* de maneira a determinar a ação a realizar para poder navegar dentro da aplicação preservando o estado atual, permitindo voltar para esses.

Cada iteração da implementação foi efetuada relativamente aos *templates* a implementar, isto é, cada iteração começa pela implementação de um novo *template*, só se concluindo depois de ser testado, corrigido e visualmente aceite pelos responsáveis na empresa. Em cada iteração as diferentes partes lógicas sofrem alterações tendo sido sempre necessário garantir a validade dos *templates* anteriormente implementados, voltando a testá-los em cada iteração.

A seguir é apresentado a arquitetura implementada no módulo sendo depois discutido a implementação das partes identificadas.

4.2.1 *Arquitetura do módulo*

O módulo segue a arquitetura MVC, tendo sido dividido em elementos visuais, modelos de dados, controladores e utilidades, sendo apresentado um diagrama do módulo na Figura 19.

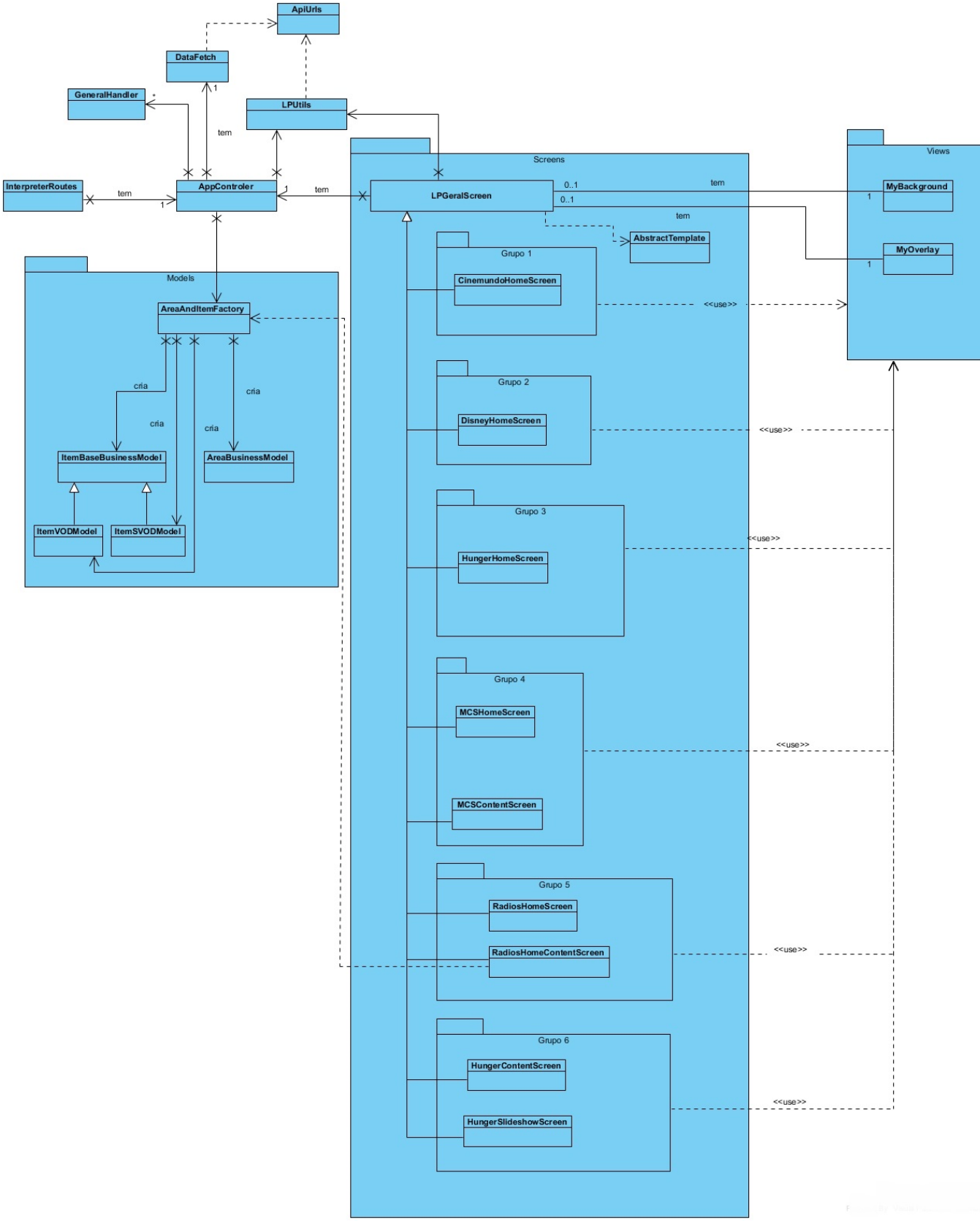


Figura 19: Diagrama de classes do modulo

Os elementos visuais incluem as *screens* e as *views*, sendo essas discutidas com mais cuidado mais a frente. As informações recuperadas da API são itens, áreas, subitens, a configuração da aplicação e a configuração dos seus *templates*. Os dados usados pelos *templates*, isto é, as áreas, itens e subitens, são representados por classes próprias, sendo que os subitens são representados como itens pois são definidos com as mesmas propriedades.

As configurações da aplicação são guardadas num objeto *javascript* no controlador. Não foi implementado uma classe específica para representar cada configuração possível porque, enquanto que um item e uma área têm um conjunto de propriedades bem definidos, as configurações variam muito de um *template* para outro. Assim, decidiu-se que era mais simples esses serem representados por um objeto simples, sendo depois passado e processado na *screen* que o usará. Os controladores são o *AppControler* e as *screens* que também servem de controladores para as suas views. O *AppControler* é responsável por recuperar e preservar os dados da API e escolher que *screen* deve ser apresentada em função do contexto que lhe é passado. Algumas classes utilitárias foram criadas, sendo a classe *LPUtils* a mais usada, permitindo aceder às configurações da API, processar a cadeia de caracteres e calcular o rácio para o redimensionamento dos elementos visuais. A classe *GeneralHandler* foi criada de maneira a poder registar *callbacks* para recuperar dados que não chegaram. É diferente do objeto *Promise*[21] geralmente usado pois essas só possuem três estados que são : em espera ou pendente, resolvida e rejeitada, não permitindo registar *callbacks* se não for criada com dados ou método. Ora é possível registar *callbacks* no *GeneralHandler* mesmo que seja criada sem argumentos, sendo os *callbacks* depois resolvidos na chegada de dados. Os dados podem ser passados através de uma *Promise*, resolvida internamente, ou diretamente. O *handler* é assim usado para guardar os dados recolhidos da API e permitir o processamento assíncrono desses.

Mais informações sobre o módulo e a sua organização são discutidos nas próximas secções.

4.2.2 *Comunicação com a API*

A *Web API* da plataforma LP possui diversos *endpoints* que recebem um número variado de parâmetros. Por isso, é usado um objeto de configuração onde são mapeados, para cada tipo de dados que pode ser recuperado, o *endpoint* da API a usar. O endpoint é especificado como uma *string* com *placeholders* que devem ser substituídos para aceder com sucesso a API. Na configuração são especificados *headers* que devem ser enviados com cada pedido e algumas regras que esses devem respeitar, nomeadamente, um tempo limite de *timeout* e um número máximo de tentativas, de maneira a não sobrecarregar o servidor. Essas regras foram implementadas no objeto *DataFetch* usado exclusivamente para efetuar pedidos à API. O controlador, através da classe anteriormente referida, efetua os pedidos e faz o *caching* dos dados recuperados. A informação é preservada até outra aplicação ser carregada, não sendo

recarregada mesmo quando sair e voltar a entrar na aplicação. Para elementos que devem ser recarregados é adicionada uma marca temporal que é verificada quando os elementos são recuperados, não sendo feito a atualização automática destes para evitar recuperar elementos não usados.

4.2.3 *Navegação no router*

A classe *InterpreterRoutes* é o router do módulo e regista uma única *route* que serve de porta de entrada para o módulo. Quando chamado, a *route* recebe um objeto que contém dados de contexto adicionado pela plataforma e dados passados quando o evento foi lançado. De maneira a poder decidir que *screen* a *route* deve devolver, foi necessário criar uma estrutura de dados passado à *route* com o contexto, que identifica, de maneira simples, o que ele deve apresentar. Esta estrutura possui duas propriedades, um número e uma *string*, chamado *id* e *target*. O *target* indica de quem é o identificador e quem deve ser carregado, e o *id* é o identificador associado ao *target*. O *router* verifica os dados passados verificando a sua validade e existência, passando depois o objeto ao controlador que irá decidir que *screen* o router deve apresentar. O controlador verifica uma segunda vez os dados passados e, usando o valor do *target*, identifica o que deve fazer. A propriedade *target* tem quatro valores válidos, *app, área, item e subitem*, que representam os diferentes elementos que podem ser os alvos de um evento de navegação. Quando não é passado um valor de *target* ou este tiver o valor *app*, o controlador verifica que o *id* é diferente daquele presente e, caso isto aconteça, limpa a sua cache e começa a carregar as configurações da nova aplicação. Uma vez as configurações carregadas o controlador recupera a configuração da *home* devolvendo-a com a *home screen* ao *router*.

Os outros valores de *target* são usados para a navegação dentro da aplicação. Por exemplo, passar um *target area* indica ao controlador que deve ir recuperar a configuração da área identificada pelo *id* e, caso exista, ver a que *template* está associado, recuperando e devolvendo a *screen* e o *template*. Caso a área não exista, é devolvido um erro. Qualquer erro, quando é feita a escolha da *screen*, implica a devolução de uma *screen* de erro com ou sem configuração, dependendo da especificação da aplicação. Uma vez com a *screen* e o seu *template*, o *router* faz a apresentação da *screen* e passa-lhe o *template* que ele irá usar. Em caso de erro, é apresentado uma *screen* de erro, pedindo ao utilizador para tentar mais tarde. Esta *screen*, não fazendo parte dos *templates* a recriar, não será apresentada neste documento.

4.2.4 *Screens e Views*

A implementação das *screens* e das *views* segue o modelo MVC, sendo que, neste contexto, as *screens* são controladores que, através do controlador geral, recupera e trata os dados,

passando-os depois às suas *views*. Assim, as *views* foram criadas de maneira a serem independentes do contexto onde são usadas, implementando unicamente lógica que permite o seu bom funcionamento.

Depois de implementar algumas *screens* verificou-se que muita lógica era repetida logo criou-se uma super classe que implementa a interface da plataforma e que seria herdada por todas as *screens* do módulo. A criação desta classe ajudou a reutilização de código e facilita correções devido a atualizações da plataforma.

Para a implementação dos *templates* de cada grupo partiu-se do princípio que esses são relativamente independentes do contexto em que foram criados, isto é, excluindo informações que eventualmente preservam no histórico e configurações que recebem, eles não sabem o contexto em que foram criados nem como os dados pedidos ao controlador são recuperados. Mesmo assim, elas possuem um contexto próprio sabendo como recuperar dados através do controlador e como os devem tratar. Isto permite restringir a implementação da *screen* ao seu contexto evitando preocupar-se com a lógica de recuperação dos dados. Além disso, a declaração dos elementos visuais foi separada das *screens* criando-se *views* auxiliares que recebem a configuração e os dados processados que depois passam aos seus filhos. Esta separação foi feita para aliviar a lógica presente nas *screens*, focando-se essa mais no processamento dos dados que recupera e ações que realiza, permitindo, também, animar os elementos visuais no seu todo sem envolver elementos da *screen* que se devem manter fixos, como o *background*.

4.2.4.1 *Views*

Foram implementadas as *views* identificadas na secção 3.2 e listadas na Tabela 1, sendo todas as *views* aqui listadas independentes da lógica do módulo. Alguns dos elementos listados na Tabela 1, depois de implementados, revelaram-se redundantes, tendo sido substituídos por outros elementos. Por exemplo, o *placeholder* presente nos elementos do carrossel foi implementado pelo *label*, pois ambos servem o mesmo propósito que é apresentar um texto sendo melhor representado pelo *label*. Decidiu-se preservar alguns elementos redundantes, como os elemento *Seta* e *Asset*, porque, mesmo que semelhantes em prático, o primeiro é usado para representar um elemento bem definido, podendo vir a mudar, enquanto que o outro apresenta uma imagem qualquer. As *views* foram implementadas de maneira a serem independentes, definindo propriedades que devem ser inicializadas pelo pai dessas e usadas para a sua estilização. Essas propriedades são definidas, por omissão, de maneira a esconder o elemento criado, evitando a aparência de elementos desproporcionados em caso de erro na configuração desses.

A seguir é apresentada uma tabela que compara os elementos identificados na Tabela 1 e o que foi implementado.

Identificado	Implementado
Label	Implementado com a *view Label* com a possibilidade de alterar a posição, o estilo de letra e o *background* do elemento.
Carrossel (sem placeholder) Carrossel (com placeholder)	Implementado com a *view Carrossel* com a possibilidade de ter ou não um *placeholder* nos seus elementos, e com a possibilidade de escolher se é ou não circular. Todas as configurações definidas originalmente são disponíveis.
Menu Vertical	Este elemento foi implementado de duas maneiras diferentes, representado pelas *views LPVerticalMenu* e *VerticalMenuWithABar*. Na primeira os elementos do menu podem possuir uma cor de fundo própria, sendo o brilho desses gerido pelo menu, enquanto que no segundo existe uma barra horizontal que movimenta-se no menu, servindo de seletor, sendo atribuído um *background* ao menu no seu todo e não aos elementos individuais. Todas as outras configurações identificadas, como o tamanho dos elementos, são suportados pelas *view*.
Seta	Implementado na *view Seta* suportando as configurações especificadas.
Contador	Implementado na *view Contador* com as funcionalidades definidas. O elemento define um padrão onde a numeração é substituído antes de apresentar o texto, sendo feito automaticamente pela *view*.
Borda	Implementado pela *view Borda* com as propriedades anteriormente definidas, sendo capaz de transicionar duma imagem para outra.
Icon	Substituído pelo elemento *asset*.
Textbox	Implementado com a *view Textbox* sendo que, além das propriedades definidas, permite definir um tamanho máximo para a apresentação do texto disponibilizando *scrolling* caso o texto não cabe no espaço definido.
Slider	Implementado na *view Slider* tendo sido adicionado uma enumeração que segue o seletor do elemento, não sendo obrigatório. A *view* implementa todas as configurações definidas.
Indicador de navegação	Este elemento foi implementado com a *view IndicadorDeNavegacao*, sendo os indicadores imagens com uma imagem indicando onde o utilizador está. A configuração da *view* respeita o que foi identificado.
Asset	Implementado pela *view* do mesmo nome com todas as configurações identificadas.

Tabela 2: Elementos identificados e a implementação efetuada

Além destes elementos também foi implementado algumas *views* adicionais, como:

- *LabelAndIcon*, que grupa e gera um *Label* e *Asset*, utilizado quando esses estiverem estritamente relacionados na UI;
- *Background*, usado para apresentar um *background* na *screen*;
- *Overlay*, usado para esconder a *screen* na sua entrada e saída, criando um efeito de transição.

4.2.4.2 *Screens*

Todas as *screens* implementadas herdam da superclasse *LPGeralScreen*, sendo que essa faz a ligação entre a definição de *screen* na plataforma e as nossas *screen*. Esta superclasse agrupa toda a lógica comum em todas as *screens* de maneira a reduzir a complexidade dos *templates* mais simples. Quando inicializadas, as *screens* recebem um objeto passado pelo router. Este contém, além de informações que eram guardadas no histórico, a configuração do *template* recuperado da API. A *screen* recupera esta configuração, guarda-a e passa-a à sua *view* primária que, depois, usa-a para configurar e estilizar as *views*. Depois da configuração passada, a *screen* começa a recuperar os dados que necessita através do controlador, processa-os e passa-os aos seus filhos que irão apresentá-los.
A *screen* recupera dados da API através de um único método do controlador, a quem passa um identificador e uma marca. Esta marca representa a profundeza para a qual a *screen* quer recuperar os dados, sendo que o identificador representa o elemento a quem os dados são associados. Esta lógica efetuado pela *screen* só depende dos dados que quer recuperar, não dependendo do estado do controlador. No controlador, a marca passada é usada para determinar que dados deve recuperar dependendo do último *target* carregado. Assim, para o mesmo *target*, duas marcas diferentes devolvem dados diferentes. A *screen* não precisa de saber donde os dados são recuperados nem como a lógica do lado do controlador é implementado, só devendo se preocupar no tratamento dos dados recuperados. Geralmente, as *screens* só efetuam um pedido ao controlador mas existem casos onde ela deve recuperar dados periodicamente, neste caso, usam um *timer* para efetuar pedidos recorrentes a API.

De maneira a dividir responsabilidades a *screen* passa os eventos de navegação as suas *view* primárias, podendo vir a realizar ações em função de como essas últimas reagem. Por exemplo, se a *view* não conseguiu realizar uma ação a *screen* pode apresentar uma mensagem ou voltar para a *screen* anterior.
A seguir são apresentadas as *screens* que implementam os grupos de *templates* identificados no Capítulo 3.2.

GRUPO 1

Como identificado na secção 3.2.1, este grupo possui um único *template*, que foi implementado com uma única *screen*. O grupo é constituído pelas classes apresentadas na Figura 20. A *screen* é associada a uma *view* que aplica a configuração e implementa alguma lógica de navegação dentro da UI. Esta *screen* só possui uma área que indica o tipo de dados com que ela deverá lidar, sendo os seus itens usados para preencher o carrossel. O texto apresentado na *textbox* assim como o comportamento do *slider* varia em função do elemento selecionado no carrossel. Esta gestão é feita na *view* principal que recupera a descrição do elemento selecionado, põe-a na *textbox*, atualiza o *slider*, o contador e outros textos quando o elemento selecionado muda.

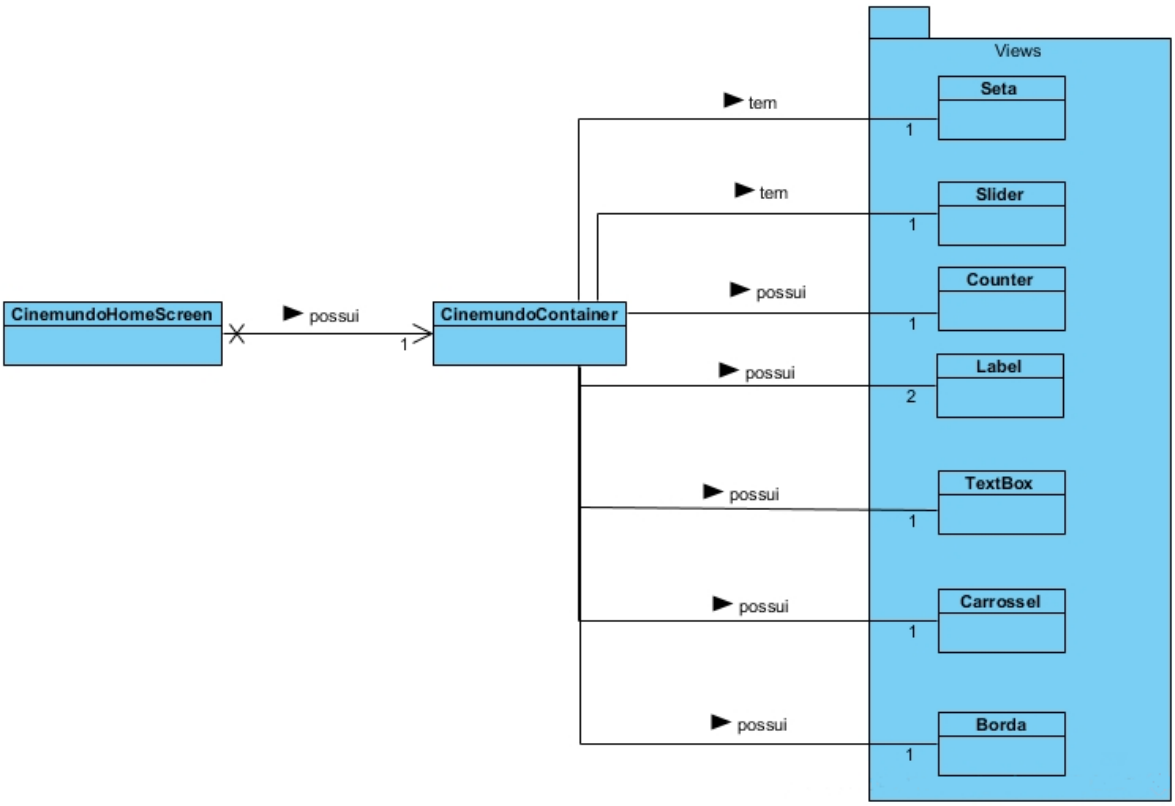


Figura 20: Elementos do Grupo 1

GRUPO 2

O grupo 2, como identificado na secção 3.2.2, possui um único *template* que é a *home screen* das aplicações deste grupo. Este *template* foi implementado por uma única *screen* conectado a uma única *view* responsável por gerir os outros elementos da UI. Na configuração pode

existir um *url* para uma *feed* de áudio que deve ser reproduzida quando carregado, por isso, é usado um leitor de áudio a quem é passado o *url* recuperado da configuração.

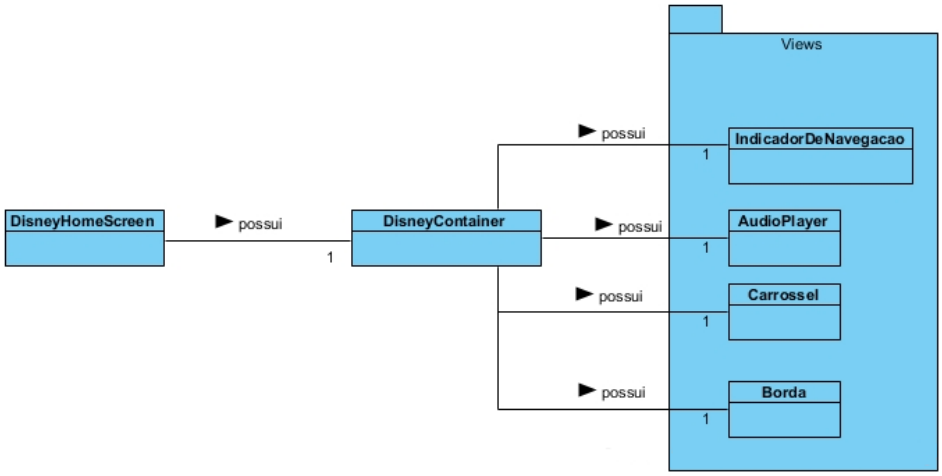


Figura 21: Elementos do Grupo 2

Esta *screen* usa as suas áreas como fonte de dados para preencher o carrossel, sendo que este pode ou não ser circular dependendo da configuração.

GRUPO 3

O grupo 3 foi implementado com uma única *screen*, apresentada na Figura 22.

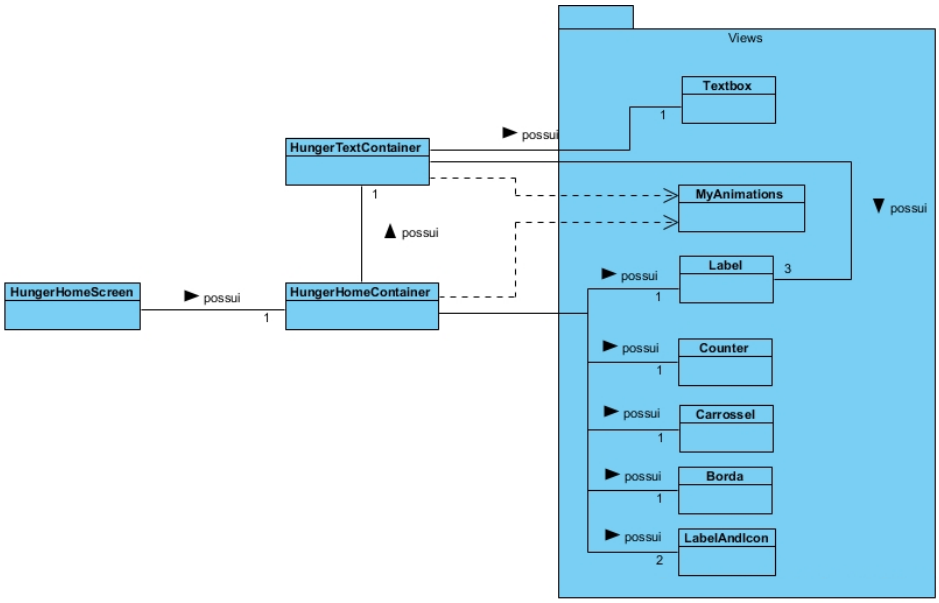


Figura 22: Elementos do Grupo 3

A UI da *screen* é controlada por uma *view* primária que atualiza os elementos. De maneira a poder implementar o segundo *template* identificado no Capítulo 3.2.3, foi criado uma *view* que esconde e apresenta os elementos do caso particular quando pedido. Assim, se a *view* primária verifica que é pedido a apresentação do caso particular, ele altera a sua configuração de maneira a sua aparência corresponder ao segundo *template*. Isto só é possível porque este só difere do *template* normal em elementos precisos, facilmente escondidos, e porque a sua configuração está especificada num único *template*, isto é, só existe um único ficheiro de configuração para ambos os casos. Este *template* representa a *home screen* das aplicações deste grupo usando as suas áreas como secções, sendo que os itens da secção selecionada são usados para preencher o carrossel. O *template* funciona como um agregador de conteúdo onde o utilizador navega entre secções sendo-lhe apresentado elementos dessa.

GRUPO 4

O grupo 4 é constituído por dois *templates* identificados na secção 3.2.4. Esses *templates* foram implementados por duas *screens*, apresentadas na Figura 23.

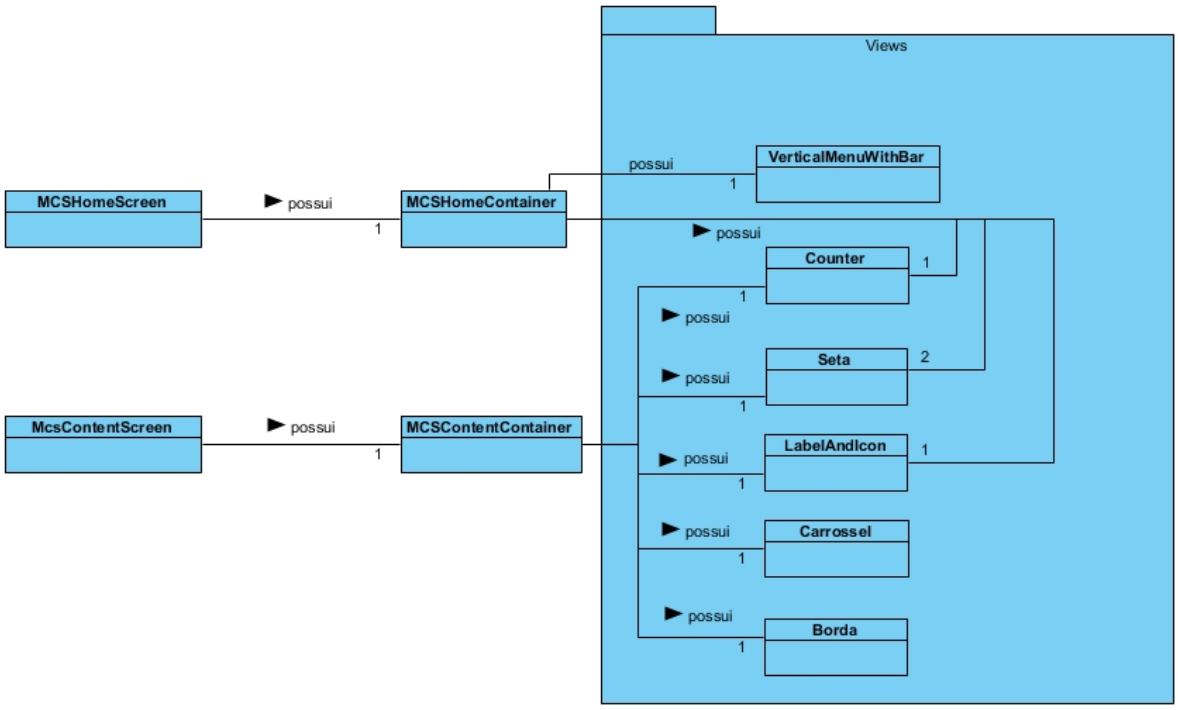


Figura 23: Elementos do Grupo 4

O primeiro *template*, a *home screen* das aplicações deste grupo, foi implementado com a *screen MCSHomeScreen* e o segundo pela *screen MCSContentScreen*. Ambas as *screens* possuem uma *view* primária responsável pela gestão da sua UI.

GRUPO 5

O grupo 5 é constituído por dois *templates* identificados na secção 3.2.5, tendo sido implementados por duas *screens*, como apresentado na Figura 24.

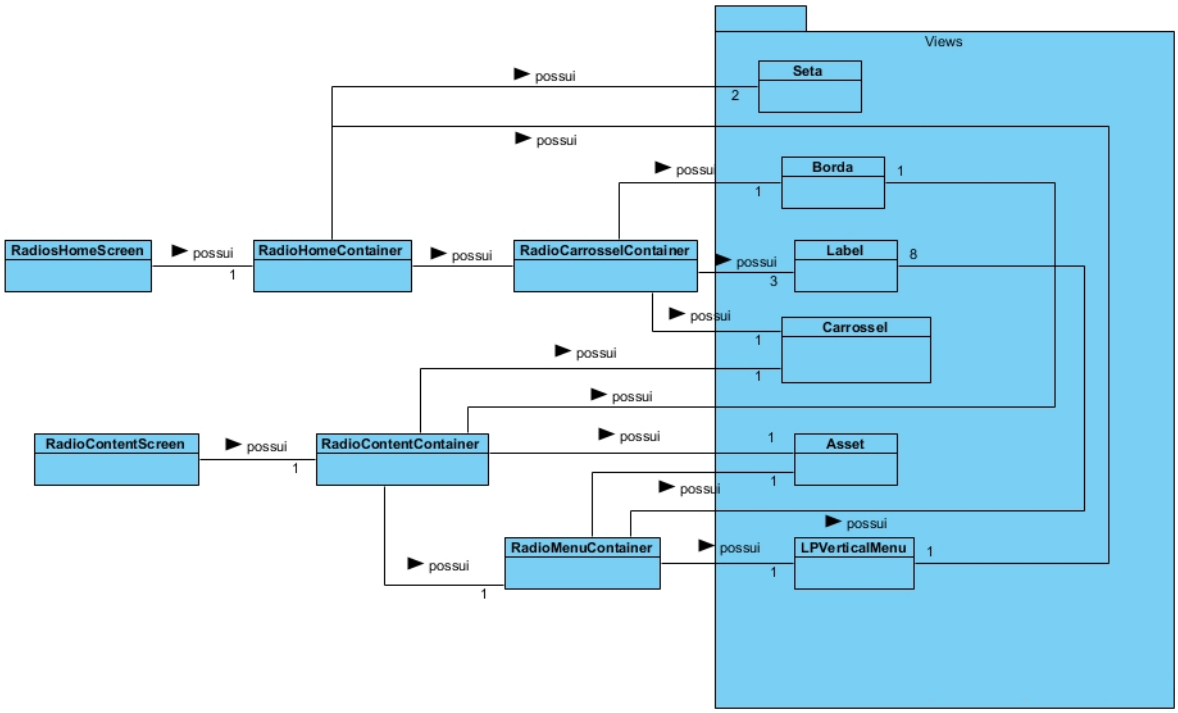


Figura 24: Elementos do Grupo 5

O primeiro *template* é implementado com a *screen RadiosHomeScreen*, sendo a sua UI representada pela sua *view* primária *RadioHomeContainer*. Neste *template* a *screen* recupera e processa as suas áreas para preencher o menu vertical. Depois, usando o elemento selecionado, a *screen* recupera e processa os *itens* associados ao elemento e passa-os a sua *view* que preenche o *carrossel* e os *labels*.

O segundo *template* foi implementado pela *screen RadiosContentScreen*, sendo a sua *UI* implementada através da *view RadioContentContainer*. Quando carregada, a *screen* recupera elementos da *api*, através do controlador, que processa para preencher o seu carrossel e preencher o menu. A seguir, a *screen* verifica se a informação do elemento selecionado tem que ser atualizada recorrentemente, indicado por uma propriedade de atraso de busca, se sim então é criado um *timer* que pede periodicamente informações ao controlador atualizando a *view* caso houver modificações.

GRUPO 6

O grupo 6 possui dois *templates*, identificados na secção 3.2.6, usados pelas aplicações dos outros grupos. A Figura 25 apresenta o diagrama de classe da implementação desse grupo.

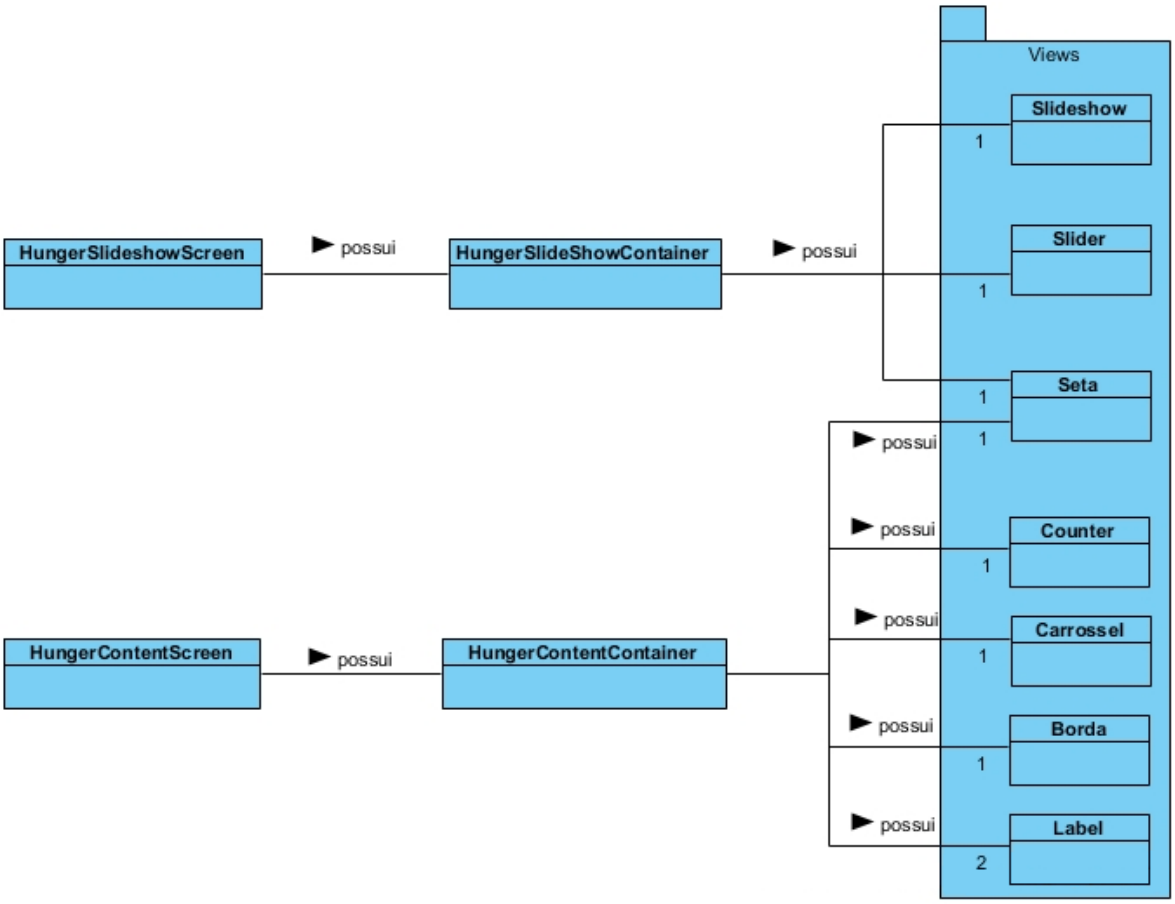


Figura 25: Elementos do Grupo 6

O primeiro *template* foi implementado na *screen HungerContentScreen* que efetua a gestão da sua *UI* através da *view HungerContentContainer*. A *UI* é composta por um carrossel preenchido por dados recuperados pelo controlador e processados pela *screen*. Neste *template* um dos *labels* é o título da *screen* e o outro é o nome do elemento selecionado, o contador seguindo a navegação do utilizador. Como definido, o carrossel é não circular sendo a seta escondida quando o primeiro elemento do carrossel não é selecionado.

O segundo *template* é mais simples e foi implementado na *screen HungerSlideshowScreen* e sua *view* primária *HungerContentContainer*. O indicador de navegação é um *slider* numerado tendo sido usado a *view Slider* e o *slideshow* foi implementado pela *view Slideshow* que, dado uma lista de *urls*, permite a apresentação sequencial de imagens. A *view* primária faz a gestão dos elementos da UI mantendo o *slideshow* e o *slider* sincronizados.

4.2.5 *Navegação nas screens*

As *screens* devem ser capazes de navegar não só para outras *screens* da aplicação como para outras aplicações e elementos da plataforma. De momento a implementação foca-se na navegação para outras aplicações e dentro dessas. Eventos de navegação acontecem unicamente quando o utilizador seleciona um elemento da *UI* ou quando decide voltar para a *screen* anterior. O último é facilmente processado através das ferramentas disponibilizadas pela plataforma, a implementação focando-se no primeiro.

Os elementos selecionados podem ser itens ou áreas, sendo que possuem diferentes tipos que implicam a realização de diferentes ações quando selecionados, como por exemplo sair da aplicação ou apresentar um vídeo, mas as ações possíveis e a lógica necessária varia se o elemento é uma área ou um item. Como a fonte de decisão é sempre o elemento decidi criar na *screen* pai, *LPGeralScreen*, um único método que, dado as informações do elemento, irá decidir que ação deve ser efetuada. Foram consideradas outras implementações, como, por exemplo, a criação de *callbacks* associadas a cada tipo de elemento que corresponderiam a ações a realizar, ou deixar o controlador, ou outra classe, decidir que ação deve ser realizada mas, no primeiro caso, o elemento é que sabe como deve ser usado, ora se isto permite facilmente modificar a implementação da ação, implica que ele conhece demasiada informação sobre o contexto onde é usado. O segundo caso é uma possível evolução do sistema implementado caso se verifica a necessidade de separar a decisão da ação a realizar da *screen* em si.

Assim, o método recebe o elemento selecionado e determina se é uma área, item ou subitem, pois possuem diferentes tipos, identificados por diferentes propriedades. A seguir é determinado, em função do tipo associado ao elemento, que ação deve ser realizado. O tipo dá informação sobre o que o elemento representa como um vídeo, uma aplicação, elementos da loja de vídeo, entro outros. Quando se move para outra *screen* dentro da plataforma, sem usar o *back*, é necessário preservar o contexto da *screen* para poder voltar a este. Para tal esta lógica foi centralizada num único método que preserva o contexto e lança o *routing* para a nova *screen*. O método foi implementado na *screen LPGeralScreen* podendo os filhos reescrever este método caso necessário.

4.3 Sumário

Como visto o desenvolvimento do interpretador foi feito iterativamente e incrementalmente, tendo-se começado com um esboço para familiarizar-se com a plataforma. Com este foi possível identificar diferentes problemas, que foram tomados em consideração durante a implementação, e elementos que seriam reusados na implementação. Depois passou-se a implementar o módulo iterando-se sob os diferentes *templates* definidos, testando e

corrigindo-os antes de passar a próxima iteração. Cada *template* foi representado por uma *screen* com os seus elementos visuais implementados com *views*. O contexto da aplicação e a comunicação é feito pelo controlador do módulo sendo abstraído das *screens* limitando a lógica presente nessas. Assim, conseguiu-se implementar todos os grupos de *templates* identificados tendo sido testados e validados.

5

CASOS DE ESTUDO E RESULTADOS

Neste capítulo são analisadas duas aplicações, uma simples e outra mais complexa, criadas na plataforma LP e que são apresentadas no *browser* usando o interpretador criado. Essas aplicações são usadas como casos de estudo para verificar a flexibilidade e robustez da plataforma de desenvolvimento. Assim, são um exemplo de aplicações que podem ser criadas na plataforma. Sendo a UI dinâmica, algumas diferenças não são visíveis nas imagens apresentadas, mas é feita uma descrição onde essas diferenças são mais notórias. Um teste de performance não foi efetuado por não ter sido possível testar a implementação no mesmo ambiente que a implementação original.

5.1 AMBIENTE DE EXECUÇÃO

A *framework* foi executado no *Google Chrome Web Browser* e na plataforma *Nw.JS*, sendo que as duas aplicações foram criadas através da plataforma *LP* e conservadas num servidor de teste donde são recuperadas.

5.2 CASOS DE ESTUDO

Para cada grupo de *templates*, excluindo o grupo seis, foram criadas aplicações no servidor LP. Usando essas aplicações, e outras já presentes, o interpretador criado foi testado, verificando que este é capaz de recuperar dados e modificar a interface para respeitar as configurações dos *templates*. Neste capítulo são apresentadas duas aplicações usadas para teste. A primeira aplicação é uma das aplicações mais simples que pode ser criada, pelo seu aspeto e suas funcionalidades, enquanto que a segunda aplicação é mais complexa tendo mais elementos visuais. Para mostrar que as configurações não são fixas, são apresentadas variações de certas páginas das aplicações.

5.2.1 *Primeira Aplicação*

A primeira aplicação foi criada com os *templates* do grupo quatro e é constituída por dois *templates* que são a página inicial e a página de conteúdos. Para tal acedeu-se ao *backend* da plataforma LP e criou-se uma nova aplicação. Depois, escolheu-se o *template* da página inicial, que aqui corresponde ao primeiro *template* do grupo, e fez-se a configuração dessa. Esta configuração implica configurar o título, o contador, o ícone, as setas e o menu vertical, sendo a configuração desta página apresentada no Anexo A. Uma vez a configuração acabada foi necessário criar áreas através do *backend*, sendo que essas irão ser usadas para preencher o menu vertical da página. A cada área é dado um nome e a identificação de um ícone a usar no menu. Nessa aplicação foram criadas seis áreas cada uma delas com um tipo. Quatro delas são associadas a *feeds RSS* externas, uma delas é uma categoria de VOD e a última possui itens próprios, sendo o tipo mais simples. Com esses dados a página inicial da aplicação é completa e visível na Figura 26.

Figura 26: Página inicial da aplicação na nova plataforma

Esta página foi configurada de maneira a que o ícone, o título e contador sejam posicionados um pouco acima a esquerda do menu vertical, sendo que este foi configurado para ocupar a parte central da página em todo a sua largura. Para além disso, o menu foi configurado para só apresentar três elementos de cada vez, os outros ficando escondidos, e as setas foram posicionadas por cima e por baixo do menu, como visível na Figura 26. No menu a barra vertical indica o elemento selecionado, subindo e descendo segundo os inputs do utilizador, e possui um efeito de brilho, passando de uma cor para outra. Nesta aplicação essa barra passa de azul para branco, mas pode ser configurada para mostrar outras cores.

A aparência da página na plataforma em uso é semelhante a aparência da página na nova *framework*, apresentado na Figura 26, como visível na Figura 27.

Figura 27: Página inicial da aplicação na plataforma hoje em uso e na nova plataforma

A página de conteúdo foi criada com o segundo *template* do grupo quatro. Para aceder a essa página o utilizador tem que selecionar um elemento do menu associado a essa. Nesta aplicação, todas as áreas foram associadas ao *template* da página de conteúdos sendo irrelevante o elemento selecionado. Como apresentado no Capítulo 3.2.4, esta página é composta por um título, um ícone, um contador, uma seta, uma borda e um carrossel, sendo a página apresentada na Figura 28

Figura 28: Página de conteúdo da aplicação

O título e o ícone foram configurados para ficarem um ao lado do outro um pouco acima do carrossel, este ficando na parte baixa da página. O título, como na página de conteúdo, é um *label* tendo sido configurado com uma cor e família de fontes de texto. A borda é configurada para não aparecer sendo que a seta só aparece quando o primeiro elemento é focado. O carrossel não é circular sendo preenchido com os itens da área selecionada. Nessa aplicação, os itens podem ser os elementos recuperados de uma das *feeds* *RSS*, itens VOD da categoria associada a área ou então itens criados no *backend* que foram associados a área. Em todos os casos a aparência dos elementos do carrossel é dinâmica, variando quando são selecionados. Aqui a configuração específica que, quando selecionado, o texto deve ter uma cor branca com fundo azul escuro, enquanto que quando não selecionados este deve ter uma cor azul claro sem fundo, como visível na Figura 28.

A aparência da página na plataforma original é semelhante mas divergem quanto ao espaço que ocupa o texto e as imagens, como visível na Figura 29.

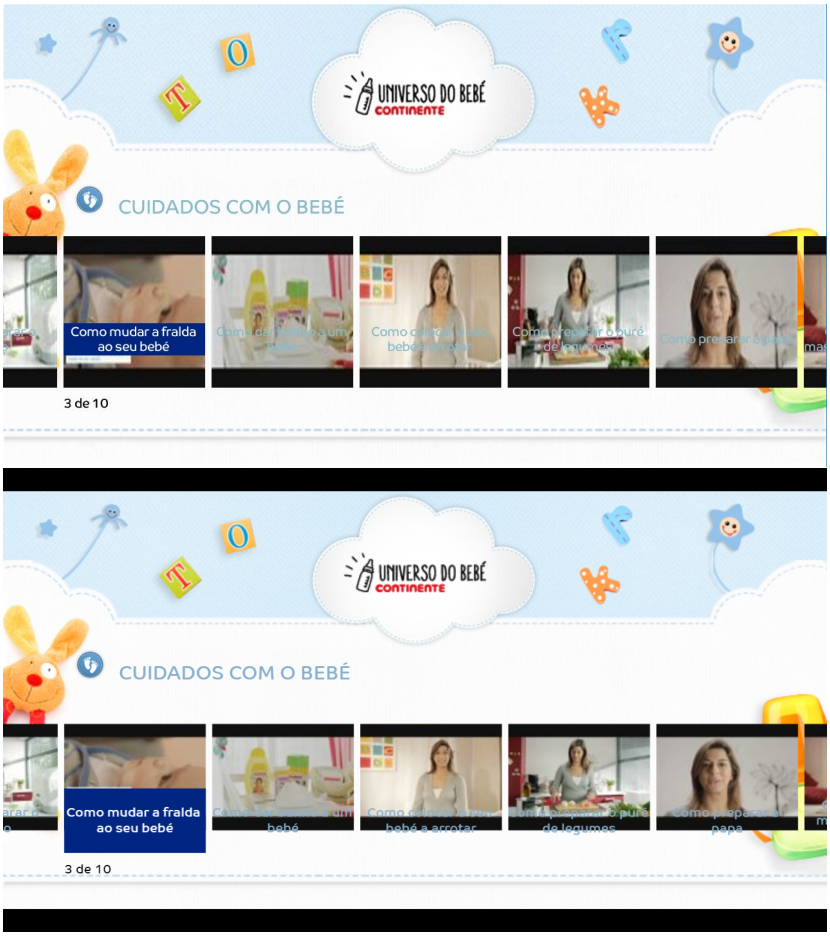


Figura 29: Página de conteúdo da aplicação na plataforma em uso e na nova plataforma

A primeira diferença deve-se ao modo como foi implementado o fundo do texto. Em vez de ser aplicado ao espaço que o texto ocupa este é aplicado ao espaço atribuído ao texto, não variando entre elementos. Os responsáveis da empresa aceitaram esta mudança. A segunda diferença deve-se a implementação da classe que permite introduzir imagens na UI pois esta só possui dois modos de redimensionamento da imagem que são *contain*, que aumenta o máximo possível a imagem mantendo sua proporção, e *cover*, que estica a imagem, mantendo a sua proporção até preencher todo o espaço. No modo *cover* as imagens eram cortadas na largura ficando desfocadas. Assim é usado o modo *contain* para que essas ficam completamente visíveis mesmo que não ocupam todo o espaço atribuído.

As configurações são facilmente alteráveis em qualquer instante através do *backend* da plataforma sendo as alterações visíveis quando se volta a aceder a aplicação, como exemplo alterou-se o aspeto da página de conteúdo da aplicação sendo o seu novo aspeto apresentado na Figura 30.

Figura 30: Página de conteúdo da aplicação modificada

5.2.2 *Segunda Aplicação*

Como para a primeira aplicação, a segunda aplicação foi criada através do *backend* da plataforma LP, sendo que em vez de usar *templates* do grupo quatro, foram usados os *templates* do grupo 5. Como visto no Capítulo 3.2.5, o grupo 5 é constituído por dois *templates*. Aqui o primeiro *template* do grupo é usado como a página inicial da aplicação. Esse *template* é complexo e constituído por vários elementos donde um menu vertical e carrossel. Para esta aplicação foram criadas oito áreas onde sete são do tipo *feed RSS* e uma do tipo saída, que faz sair o utilizador da página quando selecionada. Como na primeira aplicação, as áreas têm nomes e são associadas a outro *template* da aplicação, que no caso desta aplicação é o segundo *template* do grupo. As áreas são usadas para preencher o menu da página. Quando um elemento tem o foco o interpretador recupera os itens da área e usa-os para preencher o carrossel. Quando não existem itens, como pode acontecer em *feeds RSS*, o carrossel, a borda e os *labels* são escondidos. Com o carrossel preenchido, um dos seus elementos será focado. Quando isso acontecer, a informação do item é usada para preencher os *labels* do *template*. Em alguns casos, a informação presente no item é incompleta sendo necessário voltar a efetuar um pedido a API para obter todas as suas informações antes de preencher os *labels*. Com a configuração do *template* e a criação das suas áreas e itens, a página inicial da aplicação está acabada sendo apresentada na Figura 31 e a sua configuração é apresentada no Anexo B.

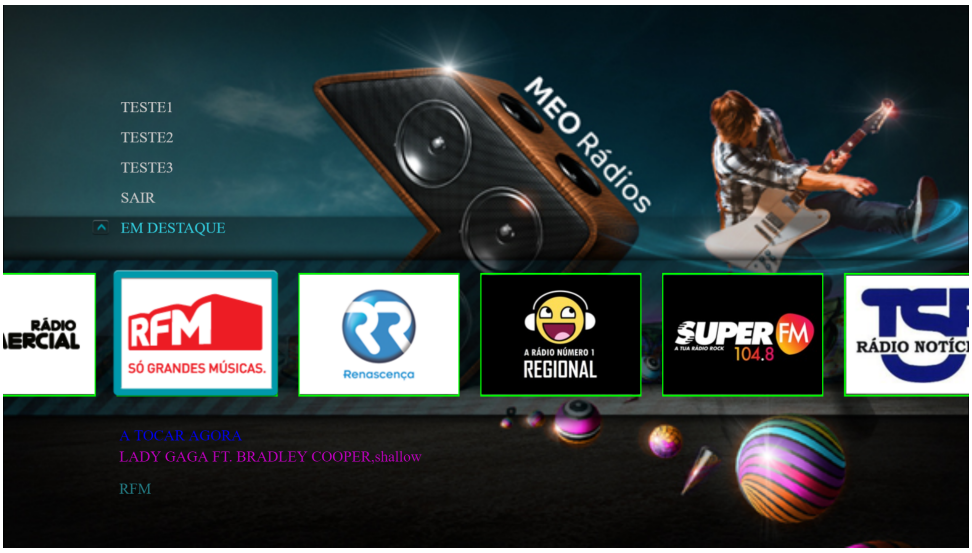


Figura 31: Página inicial da aplicação na nova plataforma

Esta página, na plataforma em uso, possui um aspeto semelhante, como é possível aperceber-se comparando a Figura 33 e a Figura 32, divergindo quanto aos elementos do menu apresentados, devendo-se a maneira como esses são implementados. Na plataforma em uso o menu é cíclico sendo que os elementos acima daquele selecionado, aqui o elemento *EM DESTAQUE*, são as últimas áreas da lista de áreas criadas, ou seja, em vez de apresentar os cincos primeiros elementos ele apresenta o primeiro e, depois, os quatros últimos de cima para baixo. Na implementação efetuada o menu não é cíclico sendo os elementos listados de cima para baixo segundo a ordem com que foram recuperados.

Figura 32: Página inicial da aplicação na plataforma em uso

O utilizador navega nas categorias através do menu e das setas cima e baixo do seu comando, e navega no conteúdo do carrossel através das setas direita e esquerda. Se carregar no botão *OK* num dos elementos, uma nova página será carregada que irá apresentar mais informações sobre o conteúdo selecionado.

A página de conteúdo foi criada, configurada e adicionada a aplicação durante a sua criação no *backend*. Uma vez criada, as áreas da aplicação, exceto uma, foram associadas a esta página. Uma vez associados, é possível aceder à página sendo essa apresentada na Figura 33.

Figura 33: Página de conteúdo da aplicação

A página usa os itens da área selecionada para preencher o seu carrossel, depois usa as informações do elemento do carrossel focado, para preencher os *labels* e o menu. Se nenhum item existir, a página ficará vazia.

O menu é preenchido usando as informações do item focado, possuindo no máximo três elementos e no mínimo um, aquele que permite sair da página. Ao conteúdo é possível associar um fluxo de áudio e outras aplicações da plataforma, por exemplo, na Figura 33 se o utilizador selecionar o elemento *EXTERNAL APP* ele será redirecionado para a aplicação associada ao conteúdo. O fluxo de áudio, se presente, fica a tocar, algo que aqui não acontece por não ser usado um leitor de áudio. Esse não é usado por se ter sido considerado desnecessário nesta fase. A borda do carrossel é constituída por duas imagens. Quando o carrossel possui o foco, o que não acontece na Figura 33, é feito a alternância entre as duas imagens da borda, recriando o efeito presente na versão original.

Como especificado no Capítulo 3.2.5, todos os elementos visuais são configuráveis e alteráveis. Por exemplo, nesta aplicação a página de conteúdo poderia ter a aparência apresentada na Figura 34, onde o menu troca de posição com o texto, sendo a cor de alguns deles modificados.

Figura 34: Página de conteúdo modificada

5.3 Discussão dos resultados obtidos

Como visto na secção anterior, dois dos seis grupos de *templates* identificados foram corretamente reproduzidos no interpretador, mostrando a capacidade deste de recuperar uma aplicação, as suas configurações, *templates* e dados, apresentando-a ao utilizador. A criação de aplicações é sempre feita no *backend* começando-se sempre com a escolha de um grupo de *templates* antes de poder configurar uma página inicial e outras páginas de conteúdo. Semelhantemente a esses dois grupos, os outros grupos também foram implementados usufruindo das mesmas propriedades em termo de usabilidade e configuração. Para além disso é de notar que, se todos os *templates* são usáveis, nem todas as funcionalidades dos originais foram implementados, seja por esses serem considerados não relevantes para esta fase, seja por falta de implementação na *framework* dos recursos necessários a esses.

5.3.1 *Divergências nas animações*

Como visto na secção 4.1.1, o motor de animação da plataforma não é maduro, não disponibilizando funcionalidades como atraso entre animações, animações de cores, entre

outras. Ora nos *templates*, é muitas vezes usado um efeito de brilho onde um elemento deve passar de uma imagem ou cor para outra, repetindo-se continuamente, como a borda do carrossel da Figura 33 e a barra horizontal do menu da Figura 28, respetivamente. Devido à falta de animações entre cores, dois elementos com as cores pretendidas são usados, sendo modificado a opacidade dos elementos de maneira a simular a passagem de uma cor para outra. Se tal animação funciona, esta não permite transições suaves criando diferenças relativamente as transições dos *templates* originais. Na animação de brilho das imagens, também são usados dois elementos passando-se de um para outro, mas a falta de atraso entre animações implica que cada imagem só aparece brevemente, enquanto que na implementação original cada imagem fica completamente visível um certo tempo antes de desaparecer. Esse problema poderá ser resolvido em novas versões da *framework* ou, então, modificando a lógica de animação usado com uma implementação própria.

5.3.2 *Funcionalidades não implementadas*

As aplicações LP possuem opções que sinaliza ao interpretador como deve tratar os seus dados e ações que deve realizar em certas condições. Por exemplo, é possível configurar uma aplicação de maneira a que pessoas não subscritas a um produto só conseguem ver parte da aplicação, pedindo, quando o utilizador realiza uma ação, a subscrição ao produto em questão. Também é possível filtrar o conteúdo segundo a idade do utilizador de maneira a esconder conteúdos não apropriados. Essas opções, entre outras, aplicam novas regras que limitam aspetos da aplicação. Para serem aplicadas é necessário conhecer o perfil do utilizador no servidor, ora a *framework* não possui, no momento da realização deste trabalho, os dados necessários para a implementação dessas funcionalidades, não tendo sidas, por isso, implementadas. Da mesma maneira a subscrição à serviços a partir das aplicações não foi implementado.
Por além dessas regras, também verificou-se que o leitor de vídeo da *framework* só aceite tipos específicos de media e não pode ser usado com as aplicações LP. Assim, conteúdos que apresentam vídeos não podem ser reproduzidos no leitor da *framework* mas, para poder implementar parte da lógica, foi criado um objeto que permite testar a reprodução de vídeos. Tal objeto não poderá ser usado mais tarde pois deve ser usado o leitor da *framework*.
A última funcionalidade não implementada é o acesso a serviços de VOD e *SVOD* através das aplicações. Se o interpretador consegue recuperar e apresentar esse tipo de conteúdo ele não é capaz de redirecionar para a loja de vídeo donde podem ser comprados e lidos, pois essa não foi integrada na *framework*.

5.4 SUMÁRIO

As duas aplicações analisadas permitem perceber que o interpretador criado consegue reproduzir aplicações LP e apresentar os seus conteúdos, como especificado pelas suas configurações. Mesmo assim, o interpretador criado não disponibiliza todas as funcionalidades do interpretador original, pois difere em certas animações nos *templates* e não permite a filtragem do conteúdo de acordo com o perfil do utilizador devido a uma falta de integração com serviços internos da empresa. Mesmo assim, o interpretador é suficientemente completo para permitir testar todas as aplicações LP na *framework*.

6

CONCLUSÃO E TRABALHO FUTURO

Neste capítulo é feita a conclusão do trabalho efetuado, refletindo sobre os objetivos que foram atingidos, os desafios ultrapassados, a abordagem usada e o que se conseguiu demonstrar.

6.1 CONCLUSÃO

Neste trabalho conseguiu-se criar um módulo, para um serviço da *Altice Labs*, numa plataforma privada em desenvolvimento e com documentação limitada. O módulo serve de interpretador para aplicações interativas LP na nova plataforma e foi escrito em JavaScript.

A criação do interpretador e a implementação dos *templates*, permitiu demonstrar que serviços e aplicações semelhantes e usados em produção na empresa, podem ser implementados na nova *framework*, mas também permitiu identificar lacunas nesta que limitam, em parte, a implementação desses serviços. Essas lacunas, podem ser corrigidas pois a *framework* pertence à empresa, não sendo limitado por um terceiro como acontece na plataforma *Mediaroom*.

A divisão dos *templates* em grupos ajudou no processo iterativo feito na implementação. Pois, como os *templates* partilham muitos elementos visuais semelhantes, acabar um grupo de *templates* reduzia o trabalho a efetuar na iteração seguinte. Mesmo assim, tiveram que ser implementados muitas *views* pois a *framework* não disponibilizava componentes que respondiam as necessidades dos *templates*.

Se a abordagem escolhida ajudou no desenvolvimento, a falta de documentação da *framework* e uma falta de conhecimento sob as funcionalidades das aplicações LP foram problemas que me acompanharam durante todo o trabalho. Com o primeiro caso, perdeu-se tempo em pesquisar no código fonte, exemplos de funcionalidades ou componentes que poderiam ter ajudado na realização deste trabalho mas acabou-se por verificar que não existiam. No segundo caso, a implementação dos *templates* foi feita, usando como fonte de informação documentos fornecidos pela empresa, mas esses documentos só especificavam os elementos visuais e não explicavam as funcionalidades dos *templates*, que dados recuperariam, como

eram tratados nem como seriam afetados por parâmetros configurados na aplicação e nos dados. Muitos desses assuntos foram resolvidos durante a implementação, mas outros só foram percebidos no fim do trabalho não tendo sido implementados. Nesta fase de teste, a empresa decidiu que essas funcionalidades não são necessárias, sendo que, serão adicionadas ao interpretador e à *framework* mais tarde. Mesmo assim, poder trabalhar numa plataforma privada em desenvolvimento foi um desafio único que permitiu compreender a necessidade de planear e organizar o trabalho a efetuar, de maneira a esse não ser desperdiçado em caso de mudanças. Este trabalho permitiu realizar a importância de ferramentas que apoiam a criação rápida de aplicações como é o caso da plataforma LP, porque reduzem o trabalho a efetuar e o tempo de desenvolvimento. Por exemplo, as duas aplicações apresentadas no Capítulo 5 foram criadas num único dia, enquanto que o desenvolvimento normal dessas aplicações diretamente na *framework* e seguindo as etapas apresentadas na secção 2.3 teria demorado semanas.

Assim, os objetivos do trabalho foram atingidos com a implementação de todos os grupos de *templates* e a possibilidade de usar as aplicações LP na *framework*, havendo a possibilidade da sua evolução futura, implementando outras funcionalidades ainda não disponíveis que tornarão toda a solução mais completa e robusta.

6.2 TRABALHO FUTURO

O interpretador deve ser mantido de maneira a acompanhar as evoluções tanto da *framework* como da plataforma *LP* com a implementação de novos *templates*. Para além desta manutenção, e como visto anteriormente na secção 5.3, existem funcionalidades das aplicações LP que não foram implementadas neste trabalho sendo assim necessário fazer a implementação dessas, continuando o desenvolvimento do interpretador. No futuro, seria também necessário rever a arquitetura do módulo, fazendo o *refactoring* desse, de maneira a melhorar o seu desempenho e reduzir a sua complexidade, pois esse foi criado com conhecimentos reduzidos da *framework* podendo vir a ser melhorado. Um foco principal desse processo deverá estar no controlador e *router* devido a lógica lá presente, pois muito do trabalho é feito no controlador devendo ser possível dividi-lo em partes mais especializadas. Por exemplo, na implementação atual o controlador faz o mapeamento entre *template* e a sua correspondente *screen* mas este processo poderia ser passado para outro componente, independente do controlador. Assim, o controlador usaria este componente para fazer o mapeamento ou esse poderia ser deixado ao router. A *framework* permite importar ficheiros de configurações, mas sem documentação não se-conseguiu usar tal ferramenta, assim recomenda-se usar essa funcionalidade para criar ficheiros de configurações para o interpretador. Esses ficheiros poderiam conter o mapeamento dos *templates* para *screens*, *urls* para a API e outras opções necessárias.

Para facilitar futuros desenvolvimentos e a manutenção do módulo, é necessário ciar uma documentação extensa da *framework* e de todos os seus componentes, com exemplos e detalhes, devendo ser revista a documentação dos serviços existentes para facilitar a aprendizagem e possíveis portes para outras plataformas desses serviços.

BIBLIOGRAPHY

[1] Smart tv market segmentation analysis report 2015. URL https://www.marketwatch.com/press-release/smart-tv-market-segmentation-application-technology-market-analysis-research-r Accessed on 2019-12-12.

[2] ANACOM. O que é uma set-top box? https://www.anacom.pt/render.jsp?contentId=928091. Accessed on 2018-12-18.

[3] T. M. A. Antunes. *Aplicação iOS para partilha, edição e controlo de UGC na TV*. PhD thesis, Faculdade de ciências e tecnologia, Universidade de Coimbra, 2 de Setembro de 2015. section 6.3.2.

[4] C. . F. a. A. C. Arthur Varushyla, 2018. URL https://www.quora.com/How-do-I-develop-apps-for-smart-TV. Acessed on 2019-07-05.

[5] K. Chorianopoulos. User interface design principles for interactive television applications. International Journal of Human-Computer Interaction, volume 24, pages 3-9.

[6] J. Clover. Mediakind puts mediafirst on legacy set-tops. https://www.broadbandtvnews.com/2018/09/06/mediafirst-puts-mediaroom-on-legacy-set-tops/, Setembro 2018. Accessed on 2018-12-18.

[7] E. Commission. Explanatory note commission recommendation on relevant product and service markets within the electronic communications sector. URL https://www.pts.se/globalassets/startpage/dokument/legala-dokument/eu-regler/explanatorynote-201410091.pdf. Accessed on 2019-10-17.

[8] S. Developers. Application development process, 2014. URL https://developer.samsung.com/tv/develop/legacy-platform-library/art00008/index. Accessed on 2019-07-05.

[9] D. DVB, ETSI. Digital video broadcasting (dvb); transport of mpeg-2 ts based dvb services over ip based networks. Accessed on 2019-01-06.

[10] ETSI. Digital video broadcasting (dvb); globally executable mhp (gem) specification 1.3 (including ott and hybrid broadcast/broadband). URL https://www.etsi.org/deliver/etsi_ts/102700_102799/102728/01.02.01_60/ts_102728v010201p.pdf. Accessed on 2019-10-17.

[11] O. I. Forum. Open iptv forum release 2 specification volume 1 - overview. http://www.oipf.tv/web-spec/volume1.html, . Accessed on 2019-10-17.

[12] O. I. Forum. Release 2 specification volume 5 - declarative application environment, . URL http://www.oipf.tv/web-spec/volume5.html. Accessed on 2019-10-17.

[13] O. I. Forum. Open iptv forum release 2 specification volume 2 - media formats, . URL http://www.oipf.tv/web-spec/volume2.html. Accessed on 2019-10-17.

[14] O. I. Forum. Open iptv forum release 2 specification volume 6 - procedural application environment, . URL http://www.oipf.tv/web-spec/volume6.html. Accessed on 2019-10-17.

[15] M. P. Inc. Video assurance and analytics best practices for ericsson mediaroom platform operators adding new on-demand and ott services. URL http://www.marinerxvu.com/wp-content/uploads/2015/03/Mariner-xVu-Mediaroom-plus-OTT-streaming-White-Paper-Q3-2016.pdf. Accessed on 2019-01-06.

[16] ITU. https://web.archive.org/web/20110916031736/http://www.itu.int/ITU-T/newslog/IPTV+Standardization+On+Track+Say+Industry+Experts.aspx. Accessed on 2018-12-22.

[17] R. J. Lopes. The design of a digital satelitte set-top box. https://web.njit.edu/~rlopes/example-2.pdf. Accessed on 2018-12-18.

[18] Luiz F.G. Soares, Marcio F. Moreno, Carlos de Salles S. Neto, and Marcelo F. Moreno. Ginga-ncl: Declarative middleware for multimedia iptv services. In *IEEE Communications Magazine (Volume: 48 , Issue: 6 , June 2010)*.

[19] D. Minoli. *IP Multicast with Applications to IPTV and Mobile DVB-H*.

[20] Mozilla Developpers Web Docs. History interface, . URL https://developer.mozilla.org/en-US/docs/Web/API/History. Accessed on 2019-04-10.

[21] Mozilla Developpers Web Docs, . URL https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. Accessed on 2019-09-20.

[22] N. OpenTV. Opentv os experience features. https://opentv.nagra.com/ux/features, . Accessed on 2018-12-11.

[23] N. OpenTV. Opentv os. https://opentv.nagra.com/os/overview-0, . Accessed on 2018-12-11.

[24] N. OpenTV. Opentv platform architecture. https://opentv.nagra.com/platform/architecture, . Accessed on 2018-12-11.

[25] N. OpenTV. Opentv os player. https://opentv.nagra.com/player, . Accessed on 2018-12-11.

[26] D. Proteste. Televisor: qual a diagonal de ecrã certa para a minha sala?, 2016. URL https://www.deco.proteste.pt/tecnologia/televisores/dicas/televisor-qual-a-diagonal-de-ecra-certa-para-a-minha-sala. Acessed on 2019-07-09.

[27] Samsung. Design. URL https://developer.samsung.com/tv/design. Accessed on 2019-10-07.

[28] T. C. Suzana Benge. Televisão digital terrestre. URL http://www.img.lx.it.pt/~fp/cav/ano2012_2013/Trabalhos_MEEC_2012_2013/Artigo9/html/content/TDT.pdf. Accessed on 2019-01-06.

[29] J. E. D. Valle and H. Perez. Module: About microsoft and digital lifestyle. URL http://download.microsoft.com/download/3/4/E/34ED7E22-516B-41F0-BD49-92CFC037B29D/msdn_About_Microsoft_Digital_Lifestyle.pdf. page 21, Accessed on 2019-01-07.

A

CONFIGURAÇÃO PÁGINA INICIAL DA APLICAÇÃO 1

–

AnimationImageId: null
BackgroundColor: ”rgba(255, 255, 255, 1)”
BackgroundImageId: ”3cc79355-4531-e911-8396-005056a8e1a9”
CounterLeft: 375
CounterPlaceholderFormat: ”–0″ –of″ –1″”
CounterTextColor: ”rgba(134, 176, 206, 1)”
CounterTextFontStyle: ”CoText28”
CounterTop: 300
CursorGlowImageId: null
CursorImageId: null
DownArrowImageHeight: 64
DownArrowImageId: ”3fc79355-4531-e911-8396-005056a8e1a9”
DownArrowImageWidth: 70
DownArrowLeft: 640
DownArrowTop: 580
GridLeft: 0
GridTop: 370
IconHeight: 57
IconImageId: ”3dc79355-4531-e911-8396-005056a8e1a9”
IconLeft: 110
IconTop: 280
IconWidth: 57
Id: 329
IsHomePage: true
ItemBackgroundColor: ”rgba(255, 255, 255, 0)”
ItemBackgroundColorGlow: ”rgba(134, 176, 206, 0.85)”
ItemBackgroundColorSelected: ”rgba(134, 176, 206, 1)”
ItemHeight: 62

ItemIconHeight: 50
ItemIconLeft: 400
ItemIconTop: 3
ItemIconWidth: 50
ItemTextColor: "rgba(134, 176, 206, 1)"
ItemTextColorSelected: "rgba(255, 255, 255, 1)"
ItemTextFontStyle: "CoText26"
ItemTextFontStyleSelected: "CoText26"
ItemTextLeft: 480
ItemTextTop: 20
ItemTextWidth: 700
ItemTypeVideoOverlayImageId: null
ItemWidth: 1280
LeftArrowImageId: null
NumberOfVisibleElements: 3
SelectorSelectedImageId: null
SelectorUnselectedImageId: null
SliderBarImageId: null
SliderSelectorImageId: null
SplashScreenImageId: null
Title: "CATEGORIAS"
TitleLeft: 180
TitleTextColor: "rgba(134, 176, 206, 1)"
TitleTextFontStyle: "CoText28"
TitleTop: 300
Type: "MCSHOME"
UpArrowImageHeight: 64
UpArrowImageId: "3ec79355-4531-e911-8396-005056a8e1a9"
UpArrowImageWidth: 70
UpArrowLeft: 640
UpArrowTop: 290
"

B

CONFIGURAÇÃO PÁGINA INICIAL DA APLICAÇÃO 2

_
AnimationImageId: null,
BackgroundColor: ”rgba(0, 0, 0, 1)”,
BackgroundImageId: ”3b80ba87-35d4-e611-8ff4-005056a8e1a9”,
CursorGlowImageId: null,
CursorHeight: 180,
CursorImageId: ”9fee27ba-e1ac-e611-95ec-005056a8e1a9”,
CursorLeft: -15,
CursorTop: -10,
CursorWidth: 240,
DownArrowImageHeight: 21,
DownArrowImageId: ”5c2773ce-e1ac-e611-95ec-005056a8e1a9”,
DownArrowImageWidth: 26,
DownArrowLeft: 117,
DownArrowTop: 308,
GridAreaItemHeight: 30,
GridAreaItemSpacing: 10,
GridAreaItemTextColor: ”rgba(200, 200, 200, 1)”,
GridAreaItemTextFontStyle: ”CoText20”,
GridAreaItemWidth: 300,
GridAreaLeft: 156,
GridAreaTop: 125,
GridLeft: 150,
GridTop: 355,
IconImageId: null,
Id: 44,
IsHomePage: true,
ItemBorderColor: ”rgba(0, 255, 0, 1)”,
ItemBorderSize: 2,

ItemHeight: 160,
ItemSelectedAreaHeight: 60,
ItemSelectedAreaWidth: 800,
ItemSelectedSubTitleAreaHeight: 30,
ItemSelectedSubTitleAreaWidth: 800,
ItemSelectedSubTitleDescAreaHeight: 30,
ItemSelectedSubTitleDescAreaWidth: 800,
ItemSelectedSubTitleDescTextColor: "rgba(200, 0, 200, 1)",
ItemSelectedSubTitleDescTextFontStyle: "CoText20",
ItemSelectedSubTitleDescTextLeft: 154,
ItemSelectedSubTitleDescTextTop: 588,
ItemSelectedSubTitleTextColor: "rgba(0, 0, 255, 1)",
ItemSelectedSubTitleTextFontStyle: "CoText20",
ItemSelectedSubTitleTextLeft: 154,
ItemSelectedSubTitleTextTop: 560,
ItemSelectedTextColor: "rgba(37, 128, 137, 1)",
ItemSelectedTextFontStyle: "CoText20",
ItemSelectedTextLeft: 154,
ItemSelectedTextTop: 630,
ItemSpacing: 27,
ItemTypeVideoOverlayImageId: null,
ItemWidth: 210,
LeftArrowImageId: null,
NumberOfVisibleElements: 5,
SelectedAreaHeight: 40,
SelectedAreaMarginTop: 6,
SelectedAreaTextColor: "rgba(51, 204, 221, 1)",
SelectedAreaTextFontStyle: "CoText20",
SelectedAreaWidth: 270,
SelectorSelectedImageId: null,
SelectorUnselectedImageId: null,
SliderBarImageId: null,
SliderSelectorImageId: null,
SplashScreenImageId: null,
Type: "RADIOSHOME",
UpArrowImageHeight: 21,
UpArrowImageId: "5b2773ce-e1ac-e611-95ec-005056a8e1a9",
UpArrowImageWidth: 26,

UpArrowLeft: 117,
UpArrowTop: 284
″

Este trabalho foi realizado no âmbito do Programa de Bolsas GENIUS financiado através de uma Bolsa de Iniciação Científica. A Bolsa de Iniciação Científica realiza-se no âmbito da autorização dada pelo Presidente do Conselho Diretivo da Fundação para a Ciência e Tecnologia ao regulamento de Bolsa de Investigação da Inova-Ria – Associação de Empresas para uma Rede de Inovação. O local de desenvolvimento realiza-se na sede da Altice Labs em Aveiro e a atividade a desenvolver insere-se no âmbito de projetos de Investigação e Desenvolvimento, em conformidade com a direção do curso da Universidade do Minho para a obtenção do grau de Mestrado, sendo esta realizada ao abrigo do Programa GENIUS, com financiamentos próprios.