



MESTRADO EM ENGENHARIA INFORMÁTICA

ESTÁGIO

RELATÓRIO FINAL

02/09/2014

Vi2ion: Aplicação Móvel para Protótipo Second Screen

Estagiário:

Elói José de Almeida Geria

egeria@student.dei.uc.pt

Orientador:

Professor Dr. Raul Barbosa

rbarbosa@dei.uc.pt

Orientador Externo:

Eng^o Bruno Coelho

bruno.coelho@altran.com



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



alTRAN

Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia
Universidade de Coimbra
Pólo II, Pinhal de Marrocos, 3030-290 Coimbra
Pinhal de Marrocos, 3030-290 Coimbra
Tel.: 239790000
Fax: 239701266
E-mail: info@dei.uc.pt
Url: <http://www.uc.pt/fctuc/dei>

Altran Portugal, SA
Av. das Forças Armadas, 125 –
3º (Edifício Open) 1600-079 Lisboa
Tel.: 210 331 600
Fax: 210 331 639
E-mail: info@altran.pt
Url: <http://www.altran.pt>

Agradecimentos

Aos meus pais a quem devo tudo, a sua confiança, a sua sabedoria, o seu apoio incondicional na minha educação.

Ao Eng.º Bruno Coelho, meu orientador externo, pela confiança depositada em mim, pelo apoio constante, pelo interesse no sucesso do meu estágio, sempre disponível apesar da agenda lotada.

Ao Professor Raul Barbosa pelo apoio e suporte ao longo do estágio.

Ao Eng.º Hélder Pinheiro pelo apoio e conselhos ao longo do estágio.

À Altran Portugal por me possibilitar a oportunidade de trabalhar numa equipa fantástica e de expandir o meu conhecimento com novas competências técnicas e pessoais.

Resumo

Na era da informação temos cada vez mais ecrãs a lutarem pela nossa atenção. Cada vez mais o *multi-tasking* entre os nossos diversos ecrãs é uma pratica comum. Facilmente dividimos a nossa atenção entre a TV e o nosso *smart device*. Este comportamento não é um fenómeno isolado, ou uma moda, é um habito de consumo real que se encontra instalado e que começa a ser observado de perto pela industria, chama-se *second-screen*.

A indústria atualmente encontra-se num ponto de viragem na forma como aborda este comportamento, em vez de lutar contra ele, apercebe-se que é mais vantajoso explorar as imensas oportunidades de negócio que este desperta.

Este estágio avança no sentido das novas oportunidades de mercado. Não se trata de retirar o destaque da TV, trata-se complementar o seu conteúdo através do ecrã secundário, de unir os dois ecrãs em que a ligação é o contexto.

Palavras Chave

Second Screen, Companion Experience, Automatic Content Recognition, Automatic Content Enrichment, Hybrid Mobile Development

Conteúdo:

1	Introdução	1
1.1	A Empresa e Equipa.....	1
1.2	Motivação.....	1
1.3	Enquadramento do Estágio e Objetivos.....	2
2	Estado da Arte	3
2.1	Definição da Atividade <i>Second Screen</i>	3
2.2	- Classificação das “ <i>Companion Experiences</i> ”	4
2.3	- Tecnologias de Sincronização.	6
2.3.1	- <i>Watermarking</i>	6
2.3.2	- Fingerprinting	7
2.3.3	- Timecoding	7
2.3.4	- Comparação dos Métodos de Sincronização	8
2.3.5	- Fornecedores de Serviços de Sincronização	9
2.4	- Fornecedores de Serviços de Conteúdos ACE (<i>Automated Content Enrichment</i>)	10
2.5	- Exemplos de Aplicações <i>Companion</i> (LandScape Actual)	11
2.5.1	- Viggie [17]	11
2.5.2	- GetGlue [18]	12
2.5.3	- Miso [19].....	12
2.5.4	- ZeeBox [20]	13
2.5.5	- Shufflr [21]	13
2.5.6	- IntoNow [22].....	13
2.5.7	- Hannibal [23]	13
2.5.8	- Walking Dead Kill Count [24]	14
2.5.9	- MeoGo [25].....	14
2.5.10	- AT&T U-verse Live TV [26]	14
2.5.11	- Fanhattam [27]	14
2.6	- Conclusões	15
3	Maquete V2.....	18
3.1	Introdução	18
3.2	Requisitos	18
3.3	– Arquitetura	20
3.3.1	- Diagrama de Sequencia:.....	20
3.3.2	- Partner Environment:.....	21
3.3.3	- Vi2ion Server Prototype Environment:	21
3.3.4	- Set-top-box (Browser Mock)	23
3.3.5	– Aplicação Móvel Vi2ion	23
3.4	Conclusão	28
4	- Maquete V3.....	29

4.1	-Requisitos	29
4.2	-Refactoring - <i>Frameworks MV*</i> (MVC, MVP, MVVM ...) [1]	30
4.2.1	- MVP	31
4.2.2	- MVVM	32
4.2.3	- Critérios de Escolha Frameworks MV* [2][3]	33
4.2.4	- Escolha efetuada: Backbone.js	35
4.3	Arquitetura	36
4.3.1	- Vi2ion Partner Environment	36
4.3.2	- Vi2ion Server Prototype Environment	37
4.3.3	- Base de Dados	38
4.3.4	- User Environment - Aplicação Segundo o Modelo MVP (Backbone.js)	39
4.3.5	- Camada <i>Model</i> :	39
4.3.6	- Camada <i>Presenter</i> :	39
4.3.7	- Camada <i>View</i> :	40
4.4	- Social (Facebook)	42
4.4.1	- Obstáculo das aplicações híbridas com o OAuth:	43
4.4.2	- Arquitetura da Vi2ion FB Proxy	44
4.5	- Vídeo OTT	47
4.5.1	- HTTP Adaptive Streaming	47
4.5.2	- Mpeg-Dash-AVC/264 e <i>player</i>	48
4.5.3	- Cliente DASH vi2ion	49
4.5.4	- Servidor DASH vi2ion	50
4.6	- Conclusão	50
5	Testes	52
5.1	Introdução	52
5.2	- Testes <i>Black-Box</i> :	52
5.2.1	- Execução Manual	52
5.2.1.1	- OTT- Mpeg-Dash DashIF Player	53
5.2.1.2	- Network Failure Handling	55
5.2.2	- Execução Automatizada:	56
5.3	- Testes <i>White-Box</i> :	59
5.3.1	Testes Unitários:	59
5.3.2	- Memory Management	61
5.4	Conclusão	63
6	Planeamento	65
6.1	Metodologia e Processos	68
6.2	Riscos	69
7	Conclusão	70
8	- Referências	73

Lista de Figuras:

Figura 1: Atividades Second Screen	3
Figura 2: Classificação das "Companion Experience's"	4
Figura 3: Fluxo de uma App de Interação de Conteúdo	5
Figura 4: Diagrama Arquitetura Alto Nível da MaqueteV2	20
Figura 5: Diagrama de Sequencia da MaqueteV2	20
Figura 6: Diagramas das Entidades da Base de Dados da MaqueteV2	21
Figura 7: Interface do Servidor para a TV	22
Figura 8: Interface do Servidor para a App	22
Figura 9: Design Mockup para a App Móvel	24
Figura 11: Diagrama de Actividade do Globalorchestrator	27
Figura 12: Diagrama de Actividade do ViewOrchestrator	28
Figura 13: Organização do Html	28
Figura 14: MVC Clássico	31
Figura 15: MVP	32
Figura 16: MVVM	32
Figura 17 :Arquitetura Alto Nivel da MaqueteV3	36
Figura 18: Interface do Servidor para a App	37
Figura 19: Diagrama de Entidades da Base de Dados da MaqueteV3	38
Figura 20: Modelo MVP utilizado na App	39
Figura 21: Backbone Presenter MainContentView	40
Figura 22: JQuery Mobile Responsive Grids	41
Figura 23: Responsive Design, Multi-Ecrãs	42
Figura 24: Interface da Facebook Proxy	44
Figura 25: Sequencia do Login	44
Figura 26: Ecrã de Login e Confirmação	45
Figura 27: Sequência de Logout	46
Figura 28: Sequência de uma chamada à Graph Api	46
Figura 29: Ecrã de Share	47
Figura 30: Esquema de um MPD (Media Presentation Description)	48
Figura 31: Scope do Mpeg Dash	48
Figura 32: Backbone Presenter OTTView	49
Figura 33: Reprodução de um vídeo Mpeg Dash	49
Figura 34: Visualização do Backoffice da Gestão de Vídeos	50
Figura 35: Diagrama de Transição de Ecrãs	56
Figura 36: Transições por Falha de Rede	56
Figura 37: Descrição da Automação dos Testes	57
Figura 38: Exemplo de Implementação de um Passo Automatizado -1	58
Figura 39: Exemplo de Implementação de um Passo Automatizado -2	58
Figura 40: Output dos Testes Automatizados	59
Figura 41: Profiling com Memory Leak	61
Figura 42: Detached Nodes no profiling com Memory Leak	62
Figura 43: Profiling sem Memory Leak	62
Figura 44: Detached Nodes no profiling sem Memory Leak	63
Figura 45: Planeamento da Defesa Intermédia 28/01/2014	66
Figura 46: Planeamento Atual Gantt	67
Figura 47: Diagrama Agile	68
Figura 48: Exemplo de uma Referência People	72
Figura 49: Exemplos de Referências	72

Lista de Tabelas:

Tabela 1: Grupos das Aplicações do Estado da Arte	15
Tabela 2: Sub-Classificação da Categoria "Conteúdo Aumentado"	16
Tabela 3: Requisitos MaqueteV2	18
Tabela 4: Nativo vs Híbrido	25
Tabela 5: Requisitos MaqueteV3	29
Tabela 6:Features das Frameworks Javascript	33
Tabela 7:Portofólio das Frameworks Javascript	34
Tabela 8: Api do Serviço de ACE da Leankr	36
Tabela 9:Testes de Aceitação Manuais aos Requisitos	52
Tabela 10:Mpeg Dash Test Case 1	54
Tabela 11:Mpeg Dash Test Case 2	54
Tabela 12:Mpeg Dash Test Case 3	55
Tabela 13:Mpeg Dash Test Case 4	55
Tabela 14: Execução dos Testes Unitários	59

Léxico Second Screen:

<i>Second Screen</i>	Atividade num pequeno ecrã (Mobile) em paralelo com o grande ecrã(TV)
<i>ACR</i>	Automatic Content Recognition: Funcionalidade que permite obter os dados de sincronização (identificador+tempo) de uma stream de média.
<i>ACE</i>	Automatic Content Enrichment: Funcionalidade que permite enriquecer com conteúdo relacionado o conteúdo original (conteúdo aumentado)
<i>Companion Experience</i>	Experiencia de Second Screen, especificamente desenhada para o efeito pelo produtor ou 3rd party
<i>EPG</i>	Electronic program guide
<i>Gamification</i>	Inserir o conceito de jogo, em algo que não tem os.
<i>Place Shifting/TV Everywhere</i>	Atividade de ver TV longe da TV principal
<i>OTT</i>	Over the Top: Entrega de conteúdos ou serviços sobre a internet, que não é controlada pelo operador de internet.
<i>Recommendation Engine</i>	Motor que consegue gerar recomendações de conteúdo baseado em heurísticas.
<i>STB</i>	Set-top box

Léxico da Ciência da Computação

<i>EPG</i>	Electronic program guide
<i>API</i>	Application Programming Interface
<i>HTTP</i>	Hypertext Transfer Protocol
<i>HTTPS</i>	Hypertext Transfer Protocol Secure
<i>IPTV</i>	Internet Protocol television
<i>MVC</i>	Model–View–Controller
<i>MVP</i>	Model-View-Presenter
<i>MVVM</i>	Model-ViewModel-Model
<i>REST</i>	Representational State Transfer
<i>SDK</i>	Software development kit
<i>SOAP</i>	Simple Object Access Protocol
<i>SVN</i>	Apache Subversion
<i>UI</i>	User Interface
<i>UX</i>	User Experience
<i>URL</i>	Uniform resource locator

1 Introdução

O presente estágio enquadra-se no contexto da disciplina de estágio do Mestrado em Engenharia Informática da Faculdade de Ciências e Tecnologias da Universidade de Coimbra, decorrendo-se na empresa Altran Portugal. Conta com a orientação do professor Dr. Raul Barbosa, e a orientação externa do Eng.º Bruno Coelho.

1.1 A Empresa e Equipa

A Altran é um grupo internacional especializado em consultadoria tecnológica e engenharia. O grupo fundado em 1982 é sediado em França, opera em 20 países, tendo como principal mercado o mercado Europeu. Os seus sectores principais de atuação são o sector Aeroespacial, Automóvel, Energia, Ferroviário, Financeiro, Saúde e Telecomunicações. A colaboração dentro de cada sector vai desde a definição do plano estratégico até à fase de produção.

Em Portugal o grupo tem presença desde 1998, tendo como sectores principais a Banca, o Sector Público e Telecomunicações. Na área de telecomunicações, relacionada com o presente estágio, a Altran Portugal conta com uma carteira de Clientes Nacionais (*Meo, Zon, Vodafone*) bem como clientes internacionais, nomeadamente a *France Televisions*, o operador de televisão de serviço público Francês. Sendo este último um potencial cliente ao produto que dá âmbito ao estágio.

Do lado da Altran Portugal a equipa técnica começando pelo *Practice Manager* é composta pelo Eng.º Bruno Coelho, por o Project Manager Eng.º João Quitério, e pelo UI/UX Designer Pedro Costa. Do lado da Altran Sophia Antipolis França a equipa representa-se pela *Practice Manager* Eng.ª Tacci Florence, e o Arquitecto Senior Eng.º Pierre.Mazoyer.

1.2 Motivação

Vivemos num mundo Multi-ecrã, em que as nossas vidas quotidianas são influenciadas por dispositivos móveis e plataformas ubíquas que nos oferecem acesso instantâneo a diversas fontes de informação.

Esta mudança de paradigma digital leva a que o *multi-tasking* ou comumente apelidado *Second Screen* seja uma prática comum na forma de como nós interagimos com os dispositivos.

Concretamente a revolução móvel aumentou significativamente o comportamento *Second Screening* onde as pessoas dividem a atenção e o tempo entre o grande ecrã e os ecrãs móveis. Este facto ganha expressão quando olhamos para o número de dispositivos móveis, e observamos que um terço do planeta possui um dispositivo móvel com uma ligação de dados de banda larga, e que se estima que o número de subscrições móveis suba para seis biliões em 2019 [1].

Este fenómeno não é novo, a Nielsen observa o comportamento *Second Screening* há quatro anos. Segundo o estudo do último trimestre de 2012, 85% dos cidadãos Norte Americanos usaram um *smart device* pelo menos uma vez por mês enquanto viam TV, e que 40% o fizeram todos os dias [2].

Um estudo de maior escala feito em 40 países, levado a cabo pela Ericsson mostrou que a atividade de *Second Screening* se estende a 67% dos telespectadores, 62% usam redes sociais enquanto

assistem à TV. Os números são impressionantes, mas também impressionante é a sua tendência crescente. [3]

Esta mudança de comportamentos despertou um “novo” mercado e novas formas de monetização, em que se estime que tenha um valor atual de 500 milhões de €, e projeta-se que atinja os 5.9 mil milhões de € em 2017, segundo as estimativas da Mesa Alliance [4], estudos paralelos também convergem para os mesmos valores. [5]. As previsões parecem otimistas, mas atentando o mercado observamos que por exemplo a aplicação Zeebox em 3 meses de vida atingiu uma quota de mercado de 150milhoes de €[6], a aplicação GetGlue juntamente com a Viggle (líderes no mercado Norte Americano) contam com mais de 5 milhões de utilizadores [7].

É com inspiração nestes casos de sucesso que a Altran decide criar uma oferta de *Second Screen* para oferecer aos seus clientes na área de multimédia, desde empresas de telecomunicações, operadores de televisão, ou mesmo a marcas comerciais.

1.3 Enquadramento do Estágio e Objetivos

O presente estágio enquadra-se na oferta de multimídia da Altran Portugal aos seus clientes nacionais e internacionais. Pretende-se criar um protótipo de uma solução de *Second Screen* para o primeiro passo da Altran neste mercado específico.

O protótipo é composto por uma sucessão de Maquetes que vão amadurecendo o conceito, e que vão ajudando a definir o rumo da Altran no sentido de captar interesse e perceber o que os clientes desejam.

Cada Maquete é difundida dentro do grupo Altran, onde as equipas de negócio se servem para interagir com os clientes em *business showcase*.

Atualmente o protótipo já vai na terceira versão. A entrada do Estagiário deu-se a meio dos desenvolvimentos da primeira maquete (Maquete V1), que serviu para ambientação ao conceito e às tecnologias que foram novidade às competências do Estagiário. O âmbito do estágio fixa-se na Maquete V2 e na Maquete V3. A colaboração do estagiário na Maquete V2 foi parcial, e na MaqueteV3 foi total. Uma maior descrição da colaboração encontra-se presente nas definições dos requisitos 3.2 e 4.1 respetivamente.

O objetivo é desenvolver uma aplicação que crie uma ligação entre o ecrã do dispositivo móvel e o ecrã principal, que crie novas experiencias ao estender o seu conteúdo com informação em contexto, não se limitando a replica-lo.

2 Estado da Arte

2.1 Definição da Atividade *Second Screen*

No que toca a definir a atividade *Second Screen*, a Decipher ^[1] encontrou diversos termos ao longo da literatura que no fundo caracterizam os mesmos conceitos. Para clarificar esta temática, os comportamentos foram sintetizados e agrupados em 3 grandes grupos:

Dual Screening: É atualmente o grupo mais abrangente de *second screening*. Este grupo cobre qualquer atividade que faz uso de um *second screen device* enquanto um utilizador assiste à TV. Neste grupo a interação é assíncrona com o conteúdo, e pode ir desde o utilizador ir ver a previsão do estado do tempo, ver notícias, mandar email, etc. Metaforicamente é a versão do séc. XXI de ver o jornal enquanto se assiste TV.

Atividade Síncrona: Este grupo abrange as atividades que o utilizador executa no *second screen device* relativamente ao programa de TV que está a assistir. A interação com o conteúdo da TV pode ser estreita, tal como entrar em debate via Twitter relativamente ao programa assistido, ou pode ser mais solta como ver a agenda dos próximos episódios de um certo programa.

Companion Experience: Este é o tipo de atividade mais específico de *second screening*. O utilizador efetua a sua experiência de *second screen* numa aplicação feita especificamente pelo operador para o conteúdo que está a ser transmitido. Esta aplicação tem como objetivo estender a experiência do utilizador dentro do contexto do programa.

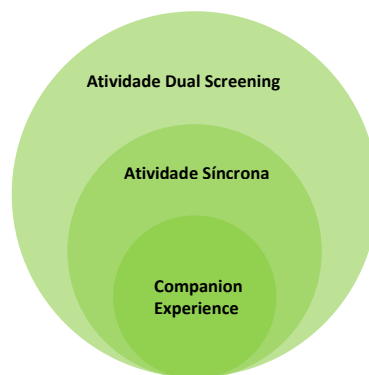


Figura 1: Atividades Second Screen

Do ponto de vista da indústria os primeiros 2 grupos são interpretados como “background-noise” em que o programa de TV tem de competir pela atenção do utilizador. É quase impossível para o operador regular ou monitorizar estes comportamentos, logo são impossíveis de rentabilizar.

Contudo o 3º grupo, o “Companion Experiences”, está sob o total controlo do operador ou serviço que controla a aplicação. Do ponto de vista do negócio é a categoria mais apetecível para os

operadores, pois permite-lhes monetizar os novos hábitos de consumo da audiência, em vez de lutar contra eles.

2.2 - Classificação das “Companion Experiences”

A análise das *Companion Experiences* feita pela Redbee/Decipher [1] pretende abranger todo o espaço passível de ser ocupado pelo mercado. Esse espaço é mapeado por 2 eixos:

Eixo “**onde/where**”: Representa o lugar onde a aplicação do *Companion Device* foi desenhada para ser usada em relação ao dispositivo de TV. I.e. *A aplicação foi feita para ser usada em frente à TV, ou foi desenhada para ser usada numa ótica de ‘place shifting’, numa outra divisão, ou mesmo fora de casa?*

Eixo “**como/how**”: Representa a funcionalidade atribuída à aplicação. A aplicação desempenha funções de discovery - I.e.: *pesquisa, recomendações, menus de informação relacionada, etc.* ou se desempenha funções de entrega de conteúdo - I.e.: *Um TV Show fornece conteúdos interativos, ou outro tipo de informação complementar relativamente ao programa.*

O resultado do cruzamento destes dois eixos apresenta uma topologia de 4 quadrantes bem definidos, onde a *Companion Experience* se pode inserir, dependendo do *onde* e *como*.

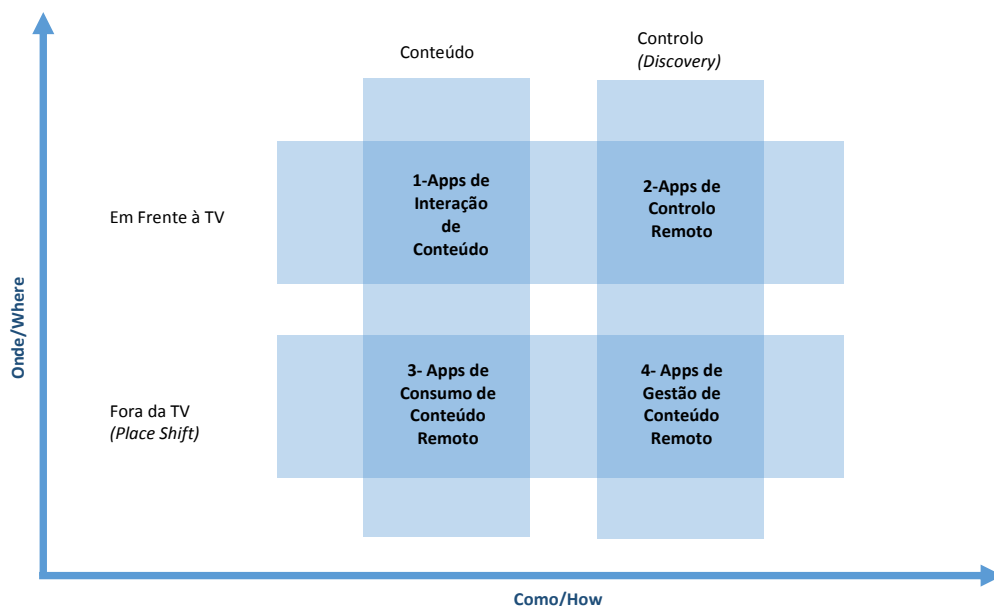


Figura 2: Classificação das "Companion Experience's"

Apps usadas “Em frente à TV”:

Interação de Conteúdo:

É a categoria mais interativa e completa a nível de *User Experience*, com uma ligação estreita com conteúdo que passa na fonte (TV).

Tipicamente uma aplicação deste quadrante segue o seguinte fluxo:

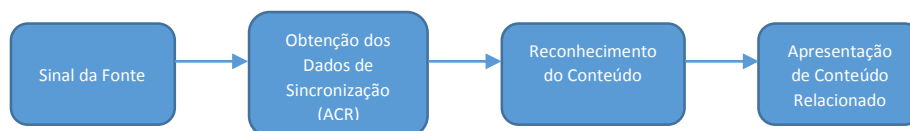


Figura 3: Fluxo de uma App de Interação de Conteúdo

A primeira etapa é através da fonte obter os dados de sincronização em que a fonte se encontra, para isto é necessário saber o identificador do vídeo da fonte, e a posição temporal em que este se encontra.

A segunda etapa é com os dados de sincronização, obter conteúdo relacionado com o conteúdo da fonte. Este processo pode ser automatizado recorrendo a técnicas de ACE (*Automatic Content Enrichment*). Finalmente os dados de reconhecimento de conteúdo são apresentados ao utilizador.

É uma zona do mercado atualmente muito verde quando comparado com as restantes categorias. Mas em contrapartida é a categoria em que se estão a assistir as maiores evoluções. E.g.: ZeeBox, Viggie, The Walking Dead Kill Count, Hannibal.

Atualmente é o quadrante que demonstra a maior tendência, e que oferece ao utilizador final a melhor experiência de utilização. É a categoria escolhida ao protótipo do vi2ion.

Remote Control:

Aplicações que foram desenhadas para substituir o comando tradicional e/ou o EPG para encontrar informações de programas, agendar gravações, recomendações, etc. E.g. Sky+ App, Samsung Remote, Zeebox.

Apps para ser usadas em ‘Place Shift’:

Consumo Remoto:

Frequentemente conhecido como *TV Everywhere*, são aplicações que dão acesso a canais, programas ou TV shows via web, quer seja live *streaming* ou conteúdo *on-demand*. E.g. ITV player, iPlayer, MeoGo.

Gestão Remota:

Apps que permitem o utilizador consultar EPG, programar gravações da sua plataforma de TV remotamente. E.g. MeoGo, Sky+

Da observação do mercado, constata-se que nenhum destes quadrantes é mutuamente exclusivo. Por exemplo a aplicação MeoGo como cliente IpTv enquadra-se no quadrante “App de consumo remoto”, e como permite consultar EPG e agendar gravações, também se enquadra como aplicação de Gestão Remota. Outro exemplo é a Zeebox, que começou no quadrante “App de Controlo Remoto” e estendeu a sua experiência oferecendo funcionalidades do quadrante “App de Interação de Conteúdo”.

2.3 - Tecnologias de Sincronização.

O principal entrave das aplicações de **Interação de Conteúdo** é a sincronização com o que se passa na fonte (TV).

Com ecossistemas de IpTv ou HbbTv é possível obter na maioria dos casos, os dados de sincronização (*TimeCoding*) diretamente da SetTopBox. Contudo estes ecossistemas embora em crescimento, não representam a globalidade do mercado, em que por exemplo nos EUA a sua taxa de penetração é da ordem dos 14% [2], e na UE de 10%, segundo o Euro-barómetro de 2013 para as telecomunicações. [3]

Nos restantes ecossistemas (TDT, Cabo, Satélite) onde está a grande maioria dos utilizadores, o *Time Coding* não está disponível, e para obter a desejada sincronização é necessário recorrer a técnicas que analisem o conteúdo em bruto da televisão para as aplicações se manterem sincronizadas. As duas principais técnicas que têm vindo a emergir são o *Audio Watermarking* e o *Áudio-Fingerprinting*.

Com a adoção por parte do mercado destas tecnologias, o sincronismo finalmente tem condições de passar a ser massificado neste tipo de aplicações.

2.3.1 - *Watermarking*

Watermarking é muitas vezes classificado como uma subclasse de *Esteganografia*, uma palavra Grega para ‘escrita escondida’. Ao contrário da criptografia, em que o objetivo é tornar uma mensagem não inteligível, a esteganografia pretende ocultar uma mensagem, embebendo-a numa outra fonte de informação. A pesquisa inicial nesta área tem origem na segunda metade dos anos 90, com um primeiro e ainda atual objetivo de defender os direitos de copyright.

A ideia base consiste em adicionar um sinal de *watermark* ao sinal original de áudio. O resultado final (sinal original + sinal *watermark*) não poderá ter diferenças percetuais para o ouvinte em relação ao sinal original.

O *Watermarking* baseia-se nos princípios da psíco-acústica, em que se toma vantagem das limitações dos modelos da perceção humana. Em particular no domínio da frequência, quando dois tons de frequências próximas são tocados ao mesmo tempo, o ouvido humano ouve apenas o tom com maior energia/amplitude, diz-se que esse tom mascara o tom com menor energia/amplitude. No domínio do temporal se dois tons estão próximos no tempo, e o primeiro tom for suficientemente alto em relação ao segundo, o primeiro mascara o segundo.

De seguida são nomeados dois modelos que exploram estes dois princípios da psíco-acústica;

Spread-Spectrum Watermarking:

Neste modelo são exploradas as propriedades de mascaramento no domínio da frequência. De um sinal de áudio $u(t)$ o modelo calcula uma máscara $Mu(f)$, chamado *masking threshold*, que é homogénea no domínio de frequência. Se o *PSD* (*power spectral density*) do sinal de *watermark* $v(t)$ for menor que o *PSD* de $Mu(f)$ para todo o domínio, então $v(t)$ é mascarado por $u(t)$. Isto significa para o ouvinte, que o sinal $v(t) + u(t)$ é perceptivelmente igual ao sinal $u(t)$.

Demonstra-se pela psíco-acústica que a inaudibilidade é conseguida com uma relação entre $u(t)$ e $v(t)$ até 20dB.[4]

Echo-Hiding Watermarking:

Neste modelo são exploradas as propriedades de mascaramento temporal. O *watermark* é um eco do sinal original, impercetível para o ouvido humano. E a distância temporal entre a onda original e a onda eco, é usada como variável para representar a informação de *watermarking*.^[4]

2.3.2 - Fingerprinting

Audio Fingerprint é um sumário digital, gerado deterministicamente a partir de um sinal de áudio. Esse sumário é elaborado pelas componentes mais relevantes no campo da percepção do som. O *Áudio Fingerprint* é atualmente um grande foco de atenção, pois veio permitir a identificação de um objeto áudio de vários megabytes, correlacionando-o com a sua *Fingerprint* de apenas alguns bytes, com a grande vantagem de ser transversal ao formato de áudio, e com uma boa resistência ao ruído do Canal.

Esta tecnologia não é novidade, mas as recentes evoluções tecnológicas, quer a nível de poder de processamento, quer a nível da algoritmia, vieram a tornar esta técnica possível de aplicar a conteúdos dinâmicos em tempo real, como uma emissão de TV em direto.

O Processo de *Fingerprinting* na sua globalidade divide-se em duas etapas ^[5]:

Construção da Base de Dados/Repositório de *Fingerprints*: A base de dados tem de possuir toda a coleção de conteúdos passíveis de identificação. Para isso cada conteúdo tem de ser processado para a obtenção da *Fingerprint*, caso seja conteúdo *Live* esse repositório tem de ser atualizado em tempo real.

Identificação do Áudio : O áudio que se deseja identificar é processado, é extraída a sua *Fingerprint*, e é comparada com as existentes na Base de Dados. De seguida o servidor faz um *matching* da *Fingerprint* com todas as *Fingerprints* existentes no repositório, em busca de um resultado positivo. Tipicamente são usadas as distâncias de Hamming para a obtenção de um grau de semelhança, o grau é comparada com um *Threshold* parametrizado dentro do algoritmo. Um algoritmo eficiente minimiza os falsos positivos e os falsos negativos.

2.3.3 - Timecoding

O Timecoding é um método generalizado de sincronização de média, que consiste em embeber no canal principal informação binária de sincronização. É da responsabilidade de cada *codec* ter uma representação deste campo, sendo que é fundamental para a apresentação do conteúdo final.

No contexto da televisão digital, o standard para a transmissão do conteúdo é o MPEG-TS (*ISO/IEC 13818-1:2007, MPEG transport stream*), que independentemente dos *codecs* utilizados para a representação de áudio e vídeo, serve de *container* para estes serem transmitidos.

Um dos campos de um pacote MPEG-TS é o PCR ^[6], acrónimo de *Program Clock Reference*. Este campo serve de referência para o cálculo do PTS – *Presentation Time Stamp*, que é o valor utilizado para a sincronização das diversas *streams* no canal, e.g: áudio, vídeo, legendas, ou uma outra qualquer *stream* existente.

Apesar de ser um standard de transmissão no *broadcast* de TV digital, apenas alguns ecossistemas permitem aceder à informação de sincronização deste protocolo. É o caso dos ecossistemas de IpTv (e.g: Microsoft MediaRoom, Cisco NDS) ou ecossistemas HbbTv, através das Api's das suas Set-top-boxes.

2.3.4 - Comparação dos Métodos de Sincronização

A maneira mais eficiente e simples de obter os dados de sincronização de uma *stream* de média, é aceder de alguma forma à meta-informação existente no seu *codec*. O protocolo de transporte MPEG-TS possui a informação necessária à sincronização com o conteúdo, contudo este método nem sempre é acessível pois grande parte do equipamento recetor de TV Digital não foi desenhado para ter interfaces que disponibilizem esta informação.

Quando tal não é possível, as técnicas de Audio Fingerprint, ou Watermark são uma alternativa. Estas duas tecnologias entre si apresentam vantagens e desvantagens.

2.3.4.1 - Vantagens Fingerprint:

-**Não modifica o sinal original**, o *Fingerprint* é construído em tempo de execução no lado do recetor a partir do sinal original. (Não há implica modificações no emissor)

No *Watermark* é necessário injetar informação extra no lado do emissor, e garantir um compromisso entre a força do sinal de *Watermark*, e a audibilidade.

-**Sinal mais robusto**, o sinal usado para a construção do *Fingerprint* são as características acústicas da própria fonte.

No *Watermark* devido ao modelo psíco-acústico a intensidade do sinal de *Watermark* é sempre muito inferior ao sinal original. A relação entre sinal original e sinal *Watermark* é na ordem dos 20dB. O que torna o *Watermark* mais frágil às alterações sofridas no canal, e também mais vulnerável a ataques por injeção de ruído.

-**Suporte conteúdo legacy**, conteúdo criado antes da construção da base de dados de *Fingerprint* é à partida bem identificado, uma vez que partilham das mesmas características acústicas. O Áudio *watermark* não possui esta característica, não é possível ter um rastreio de um conteúdo que foi emitido em que não foi inserida a *watermark*.

2.3.4.2 - Desvantagens Fingerprint :

- Para obter uma *Fingerprint* representativa do sinal, os melhores algoritmos necessitam de entre 3 a 10 segundos de recolha do sinal (algoritmo da Mufin), mais o tempo de construção da *Fingerprint* (1~2 segundos). Por outro lado, o processo de *Fingerprint* é intensivo em termos de processamento, e se estiver sempre ativo no *smartphone*, a vida da bateria é encurtada.
- O *Fingerprint* é sempre dependente de um serviço que contenha um repositório de *Fingerprints*. No caso de conteúdo live, esse repositório tem de ser atualizado em tempo real.

2.3.4.3 - Vantagens Watermarking :

- **Transporte de informação de caracter geral**, sem ter de necessariamente estar relacionada com o áudio.
- **Informação sincronizada com o áudio**, ideal para interação real-time.
- Injeção de comandos específicos, metadados de *copywrite*, Serialização de conteúdo, etc...

2.3.4.4 - Desvantagens Watermarking :

- **Canal frágil**, o sinal de *watermarking* é um sinal de baixa potencia, e pequenas alterações do canal (ruído quantização, conversão D/A, A/D, alteração taxa de amostragem, ruído e interferências do ambiente) podem originar problemas de sincronização, ou em casos extremos ilegibilidade dos dados.
- **O sinal tem de ser injetado à saída da fonte em tempo real**, implica um equipamento dedicado para o efeito.

2.3.5 - Fornecedores de Serviços de Sincronização

Um serviço de sincronização permite sincronizar um dispositivo com uma fonte de áudio genérica. O serviço ao receber uma amostra de áudio tem de ser capaz de identificar a fonte e também a posição temporal da amostra. Durante o estudo do estado da arte foram encontrados os seguintes serviços:

2.3.5.1 - Mufin [9]

A Mufin é uma empresa especializada em soluções de identificação de áudio, a sua tecnologia é baseada num motor de *áudio-fingerprint* proprietário. A identificação do áudio é transversal às diversas fontes de onde este possa surgir (e.g: TV, rádio, vídeos). A geração da *fingerprint* é feita *in-app*, ou seja, o áudio é coletado, e a *fingerprint* é gerada no próprio dispositivo, utilizando os seus recursos.

O seu produto *áudioid live channel detection* destina-se especificamente à identificação de fontes de áudio em direto, e permite às aplicações identificarem a emissão, e a posição temporal desta. Possui SDKs mobile para iOS e Android.

2.3.5.2 - Egonocast [10]

A Egonocast oferece um serviço muito semelhante ao serviço da Mufin, um serviço de identificação de áudio, transversal a qualquer fonte. A tecnologia utilizada é o *áudio-fingerprint*.

A grande diferença para o serviço da Mufin encontra-se no processo de geração da *áudio fingerprint*; No serviço da Egonocast a geração da *fingerprint* é feita do lado do servidor, a aplicação cliente limita-se a enviar a amostra de áudio, o servidor efetua o processamento necessário e responde com os dados de sincronização.

Esta solução traz vantagens e desvantagens comparativamente à geração *in-app* da *fingerprint*. Como vantagem tem-se que dispositivos com uma menor capacidade de processamento, possam utilizar o serviço sem grandes limitações, outra vantagem é o facto de a aplicação não necessitar de *updates* caso o algoritmo para a geração da *fingerprint* mude.

Como desvantagem tem-se o facto de tornar a aplicação mais sensível à latência e largura de banda da rede.

2.3.5.3 - SoundPrint [11]

O SoundPrint é um serviço da Yahoo especializado em sincronização com Tv em direto através do áudio, usando a tecnologia de áudio *fingerprint*. O seu único mercado é o mercado Norte-americano, e atualmente indexa 190 canais em tempo real.

De momento a única aplicação a utilizar o serviço é a IntoNow também da Yahoo. Não existe SDK pública, pois este serviço ainda não está a ser comercializado. Não existe uma data oficial de lançamento para o mercado.

2.3.5.4 - SyncNow ^[12]

O Syncnow é um serviço da Civolution, uma empresa com uma grande experiência na identificação de áudio com as tecnologias de áudio *fingerprint* e áudio *watermark*. O seu portfolio é maioritariamente composto por soluções de proteção de média (e.g: pré-lançamentos, cinema digital, *payTv*, etc), inteligência de média (e.g: medição de audiências de rádio e televisão). Com o advento das aplicações *2nd Screen*, a Civolution acrescentou ao seu portfolio o *Syncnow*, que oferece uma sincronização em tempo real com uma emissão em direto.

O SyncNow é baseado na tecnologia áudio *watermarking*, como tal para além da sincronização, o serviço permite injetar códigos específicos no sinal de áudio. O produto possui SDKs mobile para os dispositivos Android, iOS e Windows Phone.

2.3.5.5 - Intrasonics ^[13]

O Serviço da Intrasonics assemelha-se bastante ao serviço Syncnow da Civolution. Apresenta-se um serviço de sincronização baseado em *áudio-watermarking*, com a possibilidade de inclusão de códigos específicos no sinal de áudio.

De momento o produto apenas suporta uma SDK para iOS.

2.4 - Fornecedores de Serviços de Conteúdos ACE (*Automated Content Enrichment*)

Tipicamente os dados apresentados numa aplicação *Companion* na categoria *Interação de Conteúdo* (classificação RedBee) são pré concebidos manualmente e indexados para aparecerem num dado intervalo de tempo. Esta edição de conteúdos é feita manualmente pelas editoras, ou se for numa emissão *live* estes conteúdos são inseridos em tempo real em produção editorial.

Este processo pode ser automatizado por um serviço de ACE, esta sigla ainda não está convencionada no léxico do multimédia, pois este termo representa uma técnica que só agora está a dar os primeiros passos. Este serviço tem a capacidade de reconhecer automaticamente o conteúdo que ocorre numa *stream* de media. Conhecendo os dados de entrada: [*Identificador da stream*] e [*instante temporal*] do estado da *stream*, retorna um conjunto de metadados relacionados com o conteúdo identificado para o instante temporal pedido.

2.4.1 - Leankr ^[14]

A Leankr afirma-se como um serviço de geração automática de conteúdos que visa enriquecer a experiência de visualização de TV.

O serviço assenta na plataforma proprietária RedBox, que monitoriza em tempo real as fontes de vídeo e identifica os Metadados a cada instante para cada vídeo existente nas fontes.

Para o instante temporal pedido, a plataforma RedBox analisa o sinal de CC (*Close Caption*) existente na fonte de vídeo pedida, se este não estiver disponível, utiliza os seus algoritmos Speech to Text. Independentemente da origem do texto ser via *Close-Caption* ou de *Speech-to-Text*, o texto é analisado e filtrado. São recolhidas as palavras-chave com maior relevância semântica, e são

procurados na base de dados ou na web Metadados diretamente associados semanticamente com essas palavras-chave.

Os Metadados chegam organizados em quatro categorias: “People”, “News”, “Places” e “Products”. É ainda possível gerar *Quizes* dinamicamente para pessoas da categoria “People”.

2.4.2 - Senseeive ^[15]

A Senseeive é uma *start-up* que oferece um serviço de reconhecimento automático de conteúdo baseado em referências visuais. O serviço é baseado na tecnologia de *Image Fingerprint*, possui um dicionário cujas chaves são referências visuais, e o valor são os metadados relacionados com as referências visuais.

O Motor do serviço analisa a *stream* de vídeo pedida, no instante temporal pedido, se este encontrar alguma referência visual nesse instante, retorna os metadados correspondentes existentes no dicionário. O serviço mostra uma maior eficiência para imagens que tenham os contornos bem definidos, e.g: logotipos de marcas, capas de livros, cartazes publicitários, etc.

2.4.3 - BoxFish ^[16]

Boxfish é uma *start-up* que oferece um serviço semelhante ao da Leankr. Gera um conjunto de dados relacionados semanticamente com os dados da fonte. O serviço é alimentado pelo canal de *close caption*. Esta empresa encontra-se instalada no mercado Norte-Americano.

2.5 - Exemplos de Aplicações *Companion* (LandScape Atual)

A evolução das aplicações *Companion* foi bastante heterogenia ao longo do tempo. Aplicando a segmentação do mercado feita pela RedBee/Decipher, observa-se que os 4 quadrantes (Interação de Conteúdo, Remote Control, Consumo Remoto, Gestão Remota) não evoluíram de forma paralela.

Por exemplo os produtos do quadrante de *Consumo Remoto* e *Gestão Remota* já estão maduros, surgiram no mercado Norte-Americano há cerca de 5 anos por via das companhias **Sky** e **TiVo**, e já são uma realidade em 86% dos subscritores de TV paga nesse mercado ^[7]. Em Portugal este mercado começou a ser explorado há 3 anos pelo **MeoGo**, posteriormente surgiram as proposta do grupo Vodafone e Zon para os seus ecossistemas, e dos canais generalistas RTP, SIC e TVI numa ótica de consumo remoto das emissões em direto.

O quadrante ‘*Content Interaction*’ por outro lado é um quadrante emergente em que se observa um grande número de novas entradas.

Segundo o estudo do consórcio Second Screen Society ^[8], no final de 2012 já existiam cerca de 200 aplicações de *Companion Experience*. Para o estudo deste estado da arte foram escolhidas as que reúnem atualmente mais utilizadores, e também algumas que não tendo tantos utilizadores, oferecem funcionalidades interessantes e inovadoras.

2.5.1 - Viggle ^[17]

O Viggle é uma *companion app* com uma forte componente social, a sua utilização é baseada em *check-ins* que os utilizadores fazem relativamente ao conteúdo que estão a observar.

Os check-ins podem ser feitos manualmente, isto é, os utilizadores procuram na aplicação o programa que estão a ver, ou podem ser feitos automaticamente após a identificação automática com recurso à tecnologia de *áudio-fingerprint*.

Após a identificação do programa, o utilizador faz o *check-in* e a partir daí pode fazer interagir com as redes sociais Twitter ou Facebook, fazer *share* ou ver o que está a ser falado sobre o programa. Igualmente pode obter informações relativas ao programa no Imdb ou na Wikipedia.

Outra funcionalidade interessante é o sistema de pontos, em que o utilizador recebe pontos pelos *check-ins* feitos, por assistir a vídeos publicitários, por adicionar amigos, por partilhar nas redes sociais, entre outros. Os pontos podem ser trocados por vales de desconto em lojas aderentes, ou em serviços de TV.

Existe também uma componente de *gamification* que permite efetuar apostas (não monetárias) em eventos desportivos.

2.5.2 - GetGlue [18]

Estando no ativo desde 2008, o GetGlue é uma aplicação bem conhecida e com um grande número de utilizadores. À semelhança do Viggie, a sua utilização também se baseia em *check-in's*. A partir destes o utilizador é direcionado para um mural de comentários, fotos, vídeos, *links* entre outros, que outros utilizadores publicaram para o mesmo conteúdo.

O motor de recomendações do GetGlue é uma funcionalidade que ajuda a definir a aplicação, possui um sistema que possibilita o utilizador a classificar o conteúdo (*QuickRate*). Com este sistema de *rating* o GetGlue aperfeiçoa as recomendações de oferece ao utilizador. O conteúdo suportado vai desde uma vasta lista de canais de TV, mas também filmes e até conteúdos de clubes desportivos.

Para premiar a utilização, a aplicação oferece *stickers* que premeiam o ranking do utilizador.

2.5.3 - Miso [19]

A funcionalidade de destaque do Miso é a funcionalidade de *companion device* denominada de “*Sideshow*”. Um “*Sideshow*” tem a função de acompanhar um programa de TV, de forma síncrona ou assíncrona (a sincronização é manual, o utilizador tem de pesquisar o conteúdo), o acompanhamento é feito através de publicação de cartas com informação do que se está a ver. Estas cartas possuem ainda um formato de inquérito/votação relativamente ao conteúdo mostrado. (e.g: Que final gostaria para esta série? Opção A, B, ou C). Possuem igualmente um conjunto de ligações relacionadas com o conteúdo (e.g: Que filmes este ator já realizou, onde posso comprar uma camisola igual ao ator em cena). Cada carta está associada a um determinado momento do programa, estando dispostas em *timeline*. Este formato permite que o utilizador que esteja a consumir os conteúdos em *time-shift* / *video offline*/ *vod* possa igualmente usufruir da experiência.

A grande vantagem é que a criação das cartas é feita pela comunidade, qualquer pessoa desde um grupo de fãs, a um ator ou produtor, pode criar a sua carta de conteúdo a ser apresentada pela aplicação.

2.5.4 - ZeeBox [20]

A ZeeBox é uma *companion app* com um conjunto de funcionalidades interessantes. É possível selecionar um programa a partir de uma lista de *EPG*. Atualmente possui suporte para canais Norte Americanos, do Reino Unido e Australianos. Após ter selecionado um programa tem-se acesso a *rooms* de discussão sobre o programa, *feeds* de discussão do twitter, detalhes e informação do programa e também notícias relacionadas. É possível ver também o nível de audiência que o programa tem no momento e as *Zeetags* associadas ao programa.

As *Zeetags* são palavras-chave que se relacionam com o conteúdo, muito à semelhança das *hashtags* do *twitter*. Ao selecionar cada *Zeetag* tem-se acesso a uma compilação de informação relacionada com a *tag*, nomeadamente a uma descrição da *Wikipedia*, a uma lista de notícias relacionadas, e a uma lista de produtos com ligação a e-commerce (e.g: App's, Cd's, Livros).

Caso a set-top-box seja compatível com a aplicação (e.g: Xfinity set-top-box), a experiência de interação é estendida. Por exemplo a aplicação pode assumir funcionalidades de controlo remoto para as operações básicas (e.g: mudar de canal, controlar volume), de igual modo ao selecionar um programa na lista de *EPG* da aplicação, a set-top-box muda automaticamente de canal ou programa para o programa selecionado.

Existe ainda a secção *What's Hot*, que apresenta os conteúdos mais populares e com maior buzz.

2.5.5 - Shufflr [21]

O Shufflr é uma aplicação que compila vídeos provenientes de diferentes fontes (TV, Internet). A nível de funcionalidades esta aplicação não demonstra inovação face às restantes, destaca-se pelo design minimalista, em que as operações básicas se conseguem completar com pequenos gestos.

O ecrã principal é apelidado de "*Daily Fix*", é possível ver vídeos ou programas relacionados com o programa atual, ter acesso ao *EPG* do canal (se disponível) e através de gestos pode-se filtrar os vídeos por data, social, entretenimento, celebridades, filmes e Tv.

2.5.6 - IntoNow [22]

O IntoNow da Yahoo é uma *companion app* popular no mercado norte-americano, a única região suportada. A aplicação usa o motor *Soundprint* que monitoriza em tempo real 190 canais de televisão, servindo-se da tecnologia de *áudio-fingerprint* para a sincronização das suas aplicações. Após passada a etapa de sincronização, a aplicação faz *check-in* no seu portal que o utilizador está a assistir a um determinado conteúdo, este *check-in* pode ser estendido ao twitter ou Facebook, e direciona o utilizador para um mural dedicado ao programa a ser assistido. Este mural é composto pelas *feeds* do twitter e do Facebook.

A componente social é reforçada, pois é possível consultar os programas mais observados da lista de amigos, sendo possível também fazer-lhes recomendações.

2.5.7 - Hannibal [23]

A App *Hannibal* foi desenvolvida pela Sony Pictures exclusivamente para servir de *companion App* à série Hannibal. A aplicação sincroniza-se com a Tv com a tecnologia de *áudio fingerprint*, e uma vez sincronizada, mostra ao utilizador informação em tempo real do episódio, personagens e

cenários e de todo o universo Hannibal. Como características interessantes são a possibilidade de poder correlacionar as personagens entre si através de um diagrama em árvore, e aceder ao perfil de cada personagem. Existe também a possibilidade de consultar todos os segredos que foram progressivamente revelados até ao momento.

A componente social não foi esquecida, é possível aceder a discussões no Facebook e twitter sobre os episódios com outras pessoas que se encontrem a ver a serie.

2.5.8 - Walking Dead Kill Count [24]

A aplicação *Walking Dead Kill Count* foi desenvolvida especificamente para o universo *Walking Dead*. A aplicação está preparada para se sincronizar em qualquer episódio, com auxílio da tecnologia de *áudio watermarking*.

Esta aplicação oferece um conjunto de funcionalidades para argumentar a atenção do telespectador. Em cada episodio o utilizador pode tentar adivinhar o número de *zombies* abatidos que ira suceder, cujo resultado é comparado no final do programa.

Através do sincronismo existe um contador de *zombies* abatidos que é atualizado em tempo real. O Utilizador pode consultar perfis e um conjunto de estatísticas de cada personagem (e.g: número de zombies abatidos, armas mais usadas, armas mais eficazes). Um indicador interessante é o “*Thrill Meter*”, que é uma representação gráfica que indica o nível de excitação ou apreensão que a cena apresenta, que serve de indicador que algo irá acontecer.

Como é comum neste tipo de aplicação, existe uma componente social, em que o utilizador pode consultar e participar nas discussões do programa nas redes twitter e Facebook.

2.5.9 - MeoGo [25]

O MeoGo é uma aplicação da Meo que oferece uma experiencia completa de *place shift*. Permite aos seus clientes consumirem em qualquer local os conteúdos Meo num dispositivo móvel compatível. Permite igualmente uma gestão remota da conta meo: permite agendar gravações, e aceder ao videoclube para alugar títulos, ver trailers, ou mesmo ver títulos já alugados.

2.5.10 - AT&T U-verse Live TV [26]

AT&T U-verse Live TV é a aplicação da AT&T que oferece aos seus clientes o serviço Live TV. A sua aposta é muito focada e objetiva oferecendo aos seus clientes um conjunto de canais a que podem assistir em direto e conteúdo *on demand*. Esta aplicação é semelhante ao MeoGo.

2.5.11 - Fanhattam [27]

Assume-se como um serviço de Discovery por excelência. Permite descobrir e ver filmes, séries ou programas de TV, de uma forma em que os conteúdos são agregados dos serviços mais populares como por exemplo Hulu, Netflix, iTunes, Vudu etc. Fornece ainda informações extra sobre os conteúdos: biografias, sinopses e curiosidades. A interface é bastante acessível e apelativa, em poucos cliques o cliente está a ver o conteúdo que deseja. Está disponível para iOS e Android. Recentemente foi criada a Fan.tv, um dispositivo que se conecta à TV e à rede, e que permite usufruir das características da aplicação original no ecrã da sala.

2.6 - Conclusões

Segundo o estudo do consórcio Second Screen Society existem 5 grandes conjuntos de características que motivam o utilizador a utilizar um dispositivo para complementar a sua experiência televisiva com uma aplicação *second screen*.

2.6.1- Controlo: É a habilidade de se conseguir controlar o ecrã principal. Durante anos o dispositivo usado para esta funcionalidade foi o simples controlo remoto de infravermelhos para operações básicas (mudar canal, volume, etc), com a evolução do conceito de televisão o número de funcionalidades a controlar disparou (VOD, Agendamento de gravações, Videoclube, Epg). Uma aplicação que ofereça esta funcionalidade de forma simples e intuitiva, tem um bom argumento para captar utilizadores.

2.6.2 - Conteúdo Aumentado: É a capacidade da aplicação fornecer conteúdo complementar ao conteúdo do primeiro ecrã. Esse conteúdo pode incluir biografias de atores, estatísticas desportivas, ligações a comércio eletrónico, ou mesmo explorar a vertente de *gamification* ao incluir *Quizes*, apostas etc

2.6.3 - Social: Uma componente social em uma aplicação *Companion* permite ao utilizador expressar, discutir ou partilhar opiniões sobre o conteúdo do primeiro ecrã, permite igualmente ver o estado, ou preferências dos amigos. Este tipo de interação tipicamente é integrado com redes sociais existentes.

2.6.4 - Discovery: É a capacidade que a aplicação tem em apresentar conteúdo que responda à pesquisa ou aos gostos do utilizador. É um guia de televisão moderno, permite ao utilizador navegar na grelha televisiva e ver informação de cada programa, ou para conteúdo *on-demand* deverá ser possível encontrar o título pretendido eficazmente. Uma boa componente de *Discovery* também oferece um motor de recomendações com base no perfil do utilizador.

2.6.5 - Multi-Screen: É a capacidade da aplicação substituir o primeiro ecrã, seja para conteúdo *live* ou para conteúdo *on-demand*.

Tabela 1: Grupos das Aplicações do Estado da Arte

	Controlo	Discovery		Conteúdo Aumentado	Social	Multi-Screen
		Pesquisa	Recomendações			
Viggle				x	x	
GetGlue		x	x		x	
Miso				x	x	
ZeeBox	x	x		x	x	
Shufflr		x				x
IntoNow		x	x		x	
Hannibal				x	x	

	Controlo	Discovery		Conteúdo Aumentado	Social	Multi-Screen
Walking Dead				X	X	
MeoGo	X	X				X
U-Verse Live TV	X	X				X
Fanhattan		X	X		X	X

Devido à abrangência da categoria de “Conteúdo Aumentado”, observa-se a necessidade de a expandir. A sincronização destes pode ser feita manualmente através de “*check-ins*”, ou automaticamente através de técnicas de sincronização. O foco “Conteúdo Aumentado” pode ser específico de cada programa, ou seja é pré concebido e é preparado para ser apresentado durante o programa em instantes específicos de tempo; ou pode ser dinâmico e gerado automaticamente a partir de técnicas de ACE.

De seguida, os tipos de “Conteúdo Aumentado” são diversos e dependentes do contexto, podem ser simples definições, notícias, ligações de comércio eletrónico, estatísticas, jogos etc.

Componente de Conteúdo Aumentado:

Tabela 2: Subclassificação da Categoria "Conteúdo Aumentado"

	Tipo de Sincronização	Foco	Tipo				
			Definições	Notícias	E-commerce	Jogos (Gamification)	Outros
Viggle	AFP,CKI	Específico	X			X	
Miso	CKI	Específico					X
IntoNow	AFP	Específico			X		
ZeeBox	AFP,CKI	Genérico-ACE	X	X	X		
Hannibal	AFP	Específico					X
Walking Dead	AWM	Específico				X	X

AWM – Audio Watemark
AFP- Audio Fingerprint
CKI- Manual Check In

Do estudo efetuado constata-se que das 4 componentes (Controlo, *Discovery*, Conteúdo Aumentado e Social) as mais predominantes são a *Discovery* no campo da Pesquisa, o Conteúdo Aumentado e a componente Social. A componente de *Discovery* está bem explorada, o melhor exemplo para conteúdo televisivo encontrado foi a aplicação Zeebox que possui no separador “Guide” um *Epg* detalhado para a grande maioria dos canais Norte Americanos, com um histórico de referências até 7 dias. Para conteúdo *on-demand* a melhor aplicação encontrada foi a Fanhattan

que é um agregador de conteúdos *on-demand* dos serviços mais populares de uma forma transparente e fluida.

A componente Social observa-se que é a mais bem explorada, a grande parte das aplicações estudadas implementaram esta componente de alguma forma, maioritariamente centrado na partilha do conteúdo que o utilizador está a observar. Nesta componente destaca-se a aplicação IntoNow, que para além da convencional partilha e discussão do conteúdo nas redes sociais, a aplicação serve-se da rede social e do perfil dos seus utilizadores para gerar automaticamente recomendações.

A respeito da exploração da componente de “Conteúdo Aumentado”, observa-se que várias aplicações fizeram a sua aproximação. Numa análise mais detalhada constata-se que na grande maioria a oferta existente tem um foco específico, o conteúdo mostrado é todo pré concebido para cada programa/vídeo, e as potencialidades das técnicas de ACE ainda não estão exploradas. Com exceção da Zeebox, através das suas ZTags geradas automaticamente através do áudio.

No mercado das aplicações de *second-screen/companion* o sincronismo e o conteúdo mostrado são fatores determinantes, pelo que esta categoria ainda mostra espaço para crescer e acrescentar valor ao mercado.

A aplicação a desenvolver no âmbito deste estágio visa responder às componentes Controlo, Social e *Discovery*, mas a principal, a que se acredita trazer valor ao mercado é a componente de Conteúdo Aumentado. Será a vertente mais explorada, recorrendo a técnicas de ACE fornecidas pelo parceiro Leankr.

3 Maquete V2

3.1 Introdução

A plataforma Vi2ion pretende oferecer uma solução de *second screen* que se classifica pelos parâmetros da Decipher/RedBee como sendo uma aplicação de Interação de Conteúdo, e pelos parâmetros da Second Screen Society como sendo uma aplicação de Conteúdo Aumentado. A funcionalidade nuclear deste protótipo é a habilidade de a aplicação se sincronizar com uma emissão de TV, reconhecer o seu conteúdo de forma automática e independente do canal ou do programa, e apresentar ao utilizador um conjunto de informação relacionada com o conteúdo a que este assiste. Para este efeito o conteúdo da aplicação é enriquecido com recurso a técnicas de ACE, em que este a par com o seu sincronismo assume o destaque principal. O ACE é obtido no paradigma *SaaS* fornecido pelo parceiro Leankr. A escolha deste parceiro adequa-se ao perfil do operador *France Televisions*, que é um dos potenciais clientes deste projeto, em que todos os canais da sua grelha possuem um fluxo de *closed-caption*, que é um requerimento ao serviço de ACE da Leankr. O protótipo Vi2ion será desenvolvido de forma incremental e será composto por 2 maquetes, a maquete v2 em que o conceito se aplica a um contexto VOD, e v3 que estende a v2 com contexto *live* e novas *features*. Na MaqueteV2 o serviço de ACE será simulado com vídeos de demonstração que foram pré-processados pelo parceiro Leankr.

3.2 Requisitos

Nesta secção apresenta-se na Tabela 3 os requisitos assim como as entidades afetadas e a prioridade.

Tabela 3: Requisitos MaqueteV2

Feature type	ShortName	Impacts Tablet	Impacts Server	Impacts TV	Type	Priority
External sources	E-Commerce - link2Amazon	1	1		Functional	High
	External sources - Wiki	1	1		Functional	High
Install Package	Install Package - App install through webbrowser		1		Functional	Low
	Install Package - VM		1		Functional	High
Demo Showcase 4 BM customer Meetings	Demo Showcase 4 BM customer Meetings with internet access - Medium Case	1	1	1	Functional	High
Pairing	Pairing - by Code	1	1	1	Functional	High
	Pairing - by QR Code	1	1	1	Functional	Medium

Feature type	ShortName	Impacts Tablet	Impacts Server	Impacts TV	Type	Priority
	Pairing - More than 1 Tablet	1	1		Functional	High
	Pairing - renew sync	1			Non Functional	Medium
Audio Sync	TV-Tablet Sync	1	1		Functional	High
Auto Content Enrichment	Automatic Content Enrichment - Locations - ClosedCaptions	1	1		Functional	Medium
	Automatic Content Enrichment - News - ClosedCaptions	1	1		Functional	Medium
	Automatic Content Enrichment - People - ClosedCaptions	1	1		Functional	High
	Automatic Content Enrichment - Product - ClosedCaptions	1	1		Functional	High
TV	TV - Call2Action		1	1	Functional	Medium
	Display QR Code to Pairing			1		Medium
Tablet	Has a custom <i>timeline</i> of past events	1			Functional	High
	Typical settings must be customized	1			Functional	High
	Locally store pairing settings and reuse them when possible	1			Functional	Medium
	Multi Platform (Android, iOS)	1			Non Functional	High
	Has a <i>stream</i> mode(Allways in sync with TV)	1			Functional	High
	Has a Object mode(<i>stream</i> mode is paused and focus in a existing element in <i>timeline</i>)	1			Functional	High
	Object Mode has a timeout	1			Functional	Medium
	Uses HTTPS	1	1		Non Functional	Medium
	EPG integration	1	1		Functional	High

Após a definição dos requisitos da maquete v2 avaliou-se o peso que cada requisito tem no sucesso da maquete. Foi adicionada a coluna *Priority* que define os requisitos mínimos para o sucesso (*threshold of success*) em que para satisfazer esta condição é necessário cumprir todos os requisitos assinalados como *Priority = High*.

O diagrama UML dos casos de uso para estes requisitos encontra-se em anexo. A carga de trabalho foi distribuída entre as duas equipas Altran Portugal e Altran França. Do lado da Altran Portugal (scope do estagiário) foi definida a arquitetura para a sincronização, e implementada a Vi2ion App. Do lado da Altran França foi implementado o Servidor, e o TvPlayer (Mock da set-top-box Hbbtv).

3.3 – Arquitetura

A plataforma é composta por 3 ambientes, o ambiente do parceiro que fornece o serviço de ACE, o ambiente de *back-end* que aloja os servidores do serviço, e por fim o ambiente do utilizador final.

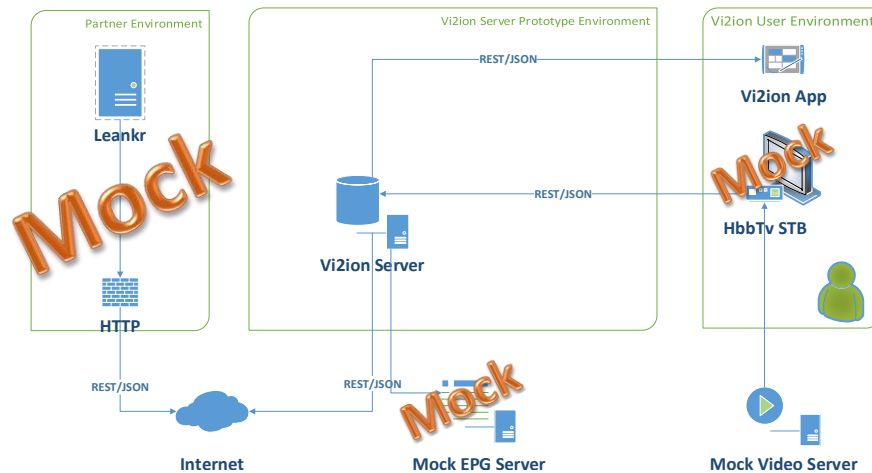


Figura 4: Diagrama Arquitetura Alto Nível da MaqueteV2

3.3.1 – Diagrama de Sequencia:

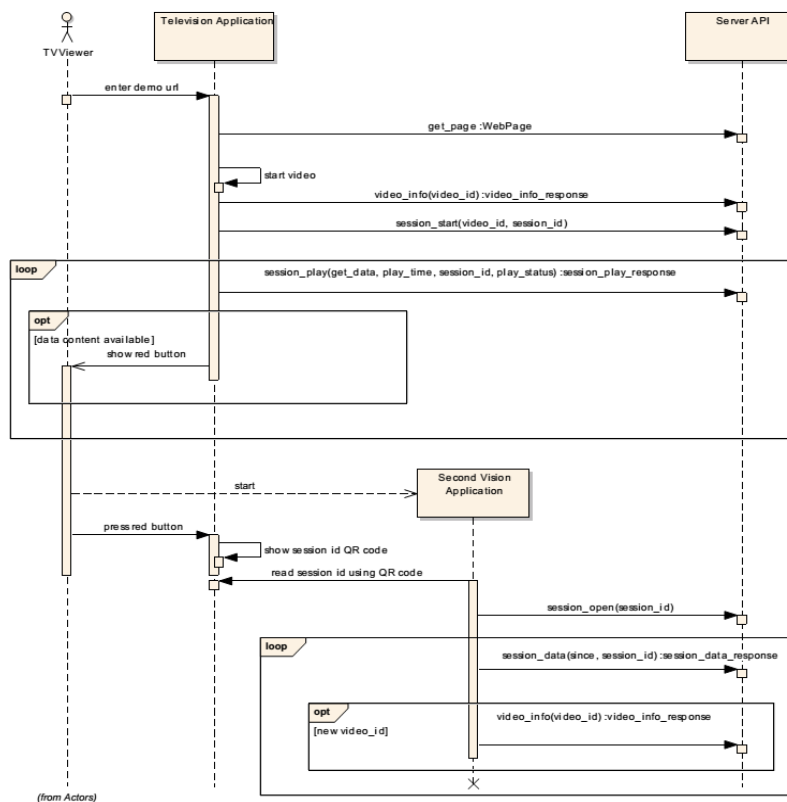


Figura 5: Diagrama de Sequencia da MaqueteV2

3.3.2 - Partner Environment:

Representa o serviço de ACE da Leankr. Esta entidade encontra-se simulada, foram cedidos pela Leankr 3 vídeos de demonstração com as respetivas referências de enriquecimento de conteúdo pré processadas. Esta informação foi guardada em base de dados no *vi2ion Server* a fim de simular a lógica do parceiro para esta MaqueteV2.

Os vídeos disponíveis são um episódio de *Democracy Now* (Notícias), um episódio de *Top Gear* (Entretenimento) e um excerto de *Now You See Me* (Filme).

3.3.3 - Vi2ion Server Prototype Environment:

O nó Vi2ion App Gateway é o core do serviço, orquestra todas as entidades da plataforma.

A tarefa principal deste nó é em primeiro lugar garantir a sincronização entre a aplicação Vi2ion App e a set-top-box alvo. Após a etapa da sincronização completa, a função seguinte é obter dados de enriquecimento de conteúdo fornecidos pelo parceiro Leankr relacionados com os dados de sincronização atuais.

Para este efeito é necessário um mecanismo de *pairing* comum entre as 3 entidades, servidor, set-top-box (*tvplayer*) e *app*, sendo o servidor o mediador.

Para este efeito existe o conceito de sessão de emparelhamento, apelidado na documentação por *session_id*. Cada set-top-box começa por registar-se no servidor, este cria um novo registo para o estado da set-top-box e associa a um *session_id* único. O *session_id* é devolvido à set-top-box para que esta possa oferecê-lo às *apps* Vi2ion que se desejem emparelhar. Esta entidade foi desenvolvida pela equipa da Altran França.

3.3.3.1 - Base de Dados

Para representar os dados de sincronização e para armazenar as referências de conteúdo aumentado é necessário uma camada de persistência que represente estes dados. A base de dados apresenta as seguintes entidades e relações:

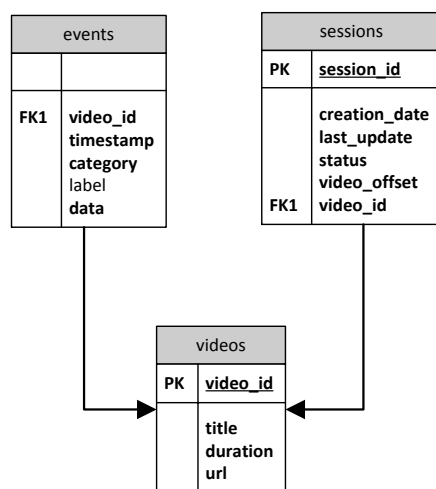


Figura 6: Diagramas das Entidades da Base de Dados da MaqueteV2

Sessions: Representa a tabela que armazena sessões, a sua informação é utilizada para emparelhar o *TvPlayer* com a aplicação. Possui uma referência para o vídeo *video_id* atual a ser reproduzido no instante *video_offset* atual na sessão *session_id*. Esta tabela é constantemente atualizada pelo Vi2ion Server a pedido do *TvPlayer* em esquema de *shortpolling*.

Vídeos: Representa a tabela que armazena a informação acerca dos vídeos disponíveis, id, duração e Url.

Events: Representa a informação relativa às referências de conteúdo enriquecido para os vídeos existentes na tabela **Vídeos**. A referência é associada ao vídeo *video_id* para o instante temporal *timestamp*, os dados da referência enquadram-se na categoria *category* (*people, news, product, places, definitions*), possui o título *label* e os dados *data*.

3.3.3.2 - API do Servidor

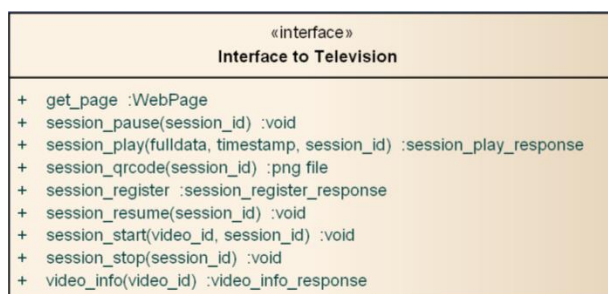


Figura 7: Interface do Servidor para a TV

+get_page(): Devolve uma página web com as *streams* de vídeo disponíveis.

+session_pause(session_id): Pausa a *stream* ativa associada à sessão *session_id*.

+session_play(fulldata, timestamp, session_id): Atualiza o estado da sessão associada ao *session_id* no servidor com o *timestamp* atual.

+session_qrcode(session_id): Devolve um binário *png* de um *qr-code* com o *session_id*. Método auxiliar usado para emparelhar a Vi2ion App.

+session_register: Regista a TV no servidor, retorna um *session_id* para esse registo.

+session_resume(session_id): Retorna a reprodução da *stream* que foi pausada, associada ao *session_id*.

+session_start(vídeo_id,session_id): Altera o vídeo atual no estado *session_id* por um novo *video_id*.

+session_stop(session_id): Muda o estado de reprodução da sessão *session_id* para stop, e coloca o seu *timestamp* a 0.

+vídeo_info(vídeo_id): Retorna informação do vídeo (simulação de *EPG* da *stream* atual)

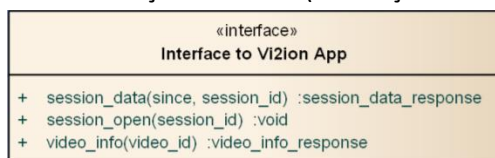


Figura 8: Interface do Servidor para a App

+session_open(session_id): Sincroniza a App com o servidor na sessão *session_id*.

+session_data(since, session_id): Obtém os dados de enriquecimento de conteúdo sincronizados com a *stream* presente na sessão *session_id*, desde o instante *since*. O campo *since* representa o *offset* da *stream* em que foram coletados os últimos dados de enriquecimento, se este valor for 0, retorna todos os dados desde o início da *stream*. Este campo é representado em UTC.

+vídeo_info(vídeo_id): Retorna informação do vídeo (simulação de *EPG* da *stream* atual).

3.3.4 - Set-top-box (Browser Mock)

A componente set-top-box é a fonte de media com que a aplicação móvel se vai sincronizar e oferecer a experiencia de enriquecimento de conteúdo. Esta set-top-box insere-se no ecossistema Hbbtv da *France Televisions*. Para efeitos de maquete, este serviço é implementado em HTML/Javascript, mas que replica o comportamento real da set-top-box HbbTv. Esta entidade começa por se registar no servidor, assim que o servidor cria uma sessão para este *player*, este começa a fazer o update do seu estado para o servidor em esquema de *short polling*. Esta entidade foi desenvolvida pela equipa da Altran França.

3.3.5 – Aplicação Móvel Vi2ion

A componente da aplicação móvel vi2ion é uma parte fundamental deste protótipo, pois é a componente que faz a ligação do serviço com o utilizador. Sendo este um protótipo de *business show case* é fundamental que tenha um design gráfico limpo e intuitivo que transmita de forma clara o conceito da aplicação, e um design funcional robusto que transmita confiança aos potenciais clientes.

A aplicação destina-se a *tablets* preferencialmente com ecrãs superiores a 7 polegadas, agnosticamente à plataforma.

3.3.5.1 – Design UI/UX

O Design gráfico assim como a experiencia de utilização são componentes fundamentais de qualquer projeto de Software destinado ao público. Esta componente tem de fazer a ligação entre o utilizador os requisitos funcionais estabelecidos de forma simples e objetiva, sem fugir do foco da aplicação.

Neste caso pretende-se um Design que demonstre as potencialidades da geração automática de conteúdo para as 4 categorias definidas pelo parceiro (*People, News, Products, Definitions e Places*), e também que consiga transmitir a noção de sincronismo com a televisão através do conceito de *timeline*.

Chegou-se então ao *mockup* da figura, que serviu de base para o design final :

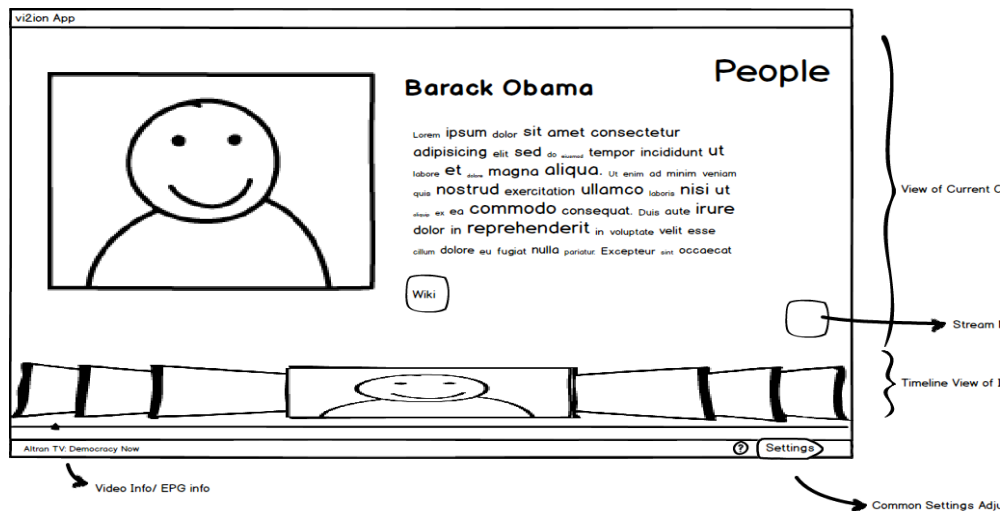


Figura 9: Design Mockup para a App Móvel

O Ecrã principal é composto maioritariamente por uma zona de foco a um determinado objeto. Esse objeto pode ser do tipo *People*, *News*, *Products* ou *Places*, sendo que cada tipo tem o seu próprio *layout*.

Para o conceito de *timeline* existe no mesmo ecrã principal um *slider* com uma representação em miniatura dos objetos identificados, ordenados por ordem de aparecimento (à esquerda está o objeto mais atual).

A nível de UX a aplicação tem dois modos, um modo “*stream*” em que está sempre a ser renderizado o objeto mais atual, e um modo “*object*”, em que a aplicação pausa o modo “*stream*” para renderizar um determinado objeto existente na *timeline*. É possível “deslizar” o *slider* e clicar em qualquer miniatura, ao clicar a aplicação comuta do estado “*stream*” para o modo “*object*”.

O campo da usabilidade e experiencia de utilização é um tema bem estudado na literatura, para a elaboração da UI/UX do protótipo foram usadas as heurísticas elaboradas pelo grupo Norman Nielsen [28].

3.3.5.2 – Multi-Plataforma

Um requisito importante é o da aplicação ser suportada por *tablets* de qualquer plataforma móvel. Este requisito levanta diversas questões e desafios. Uma forma de contornar a barreira da plataforma, é desenvolver a aplicação baseada em tecnologias web, e posteriormente fazer a conversão para uma aplicação nativa através de um *middleware*. Esta solução é conhecida como solução híbrida, pois enquadra-se entre uma aplicação web, e uma aplicação nativa.

Levanta-se a questão das aplicações nativas *versus* as aplicações web híbridas, em que se sumarizam as vantagens na seguinte tabela (em contexto com os requisitos):

Forças:

Tabela 4: Nativo vs Híbrido

Nativo	Híbrido
Melhor performance e Responsividade	Maior flexibilidade, código comum para todas as plataformas
SDK estandardizado	Maior rapidez de desenvolvimento
Total controlo dos dispositivos de sistema disponíveis	Manutenção do código mais flexível

O modelo escolhido foi o híbrido, pois os requisitos estabelecidos não requerem um poder de processamento elevado, e para as componentes de UI/UX o modelo híbrido consegue responder com fluidez. Contudo o principal fator é o facto de a plataforma ser um protótipo, onde o que interessa é transmitir o conceito aos clientes, o tempo de desenvolvimento e os recursos necessários para o desenvolvimento. Por este motivo a escolha recaiu para uma solução híbrida.

Existem SDK que materializam este modelo, exemplo do Appcelerator Titanium, Telerik Appbuilder, Intel XDK e Phonegap. Todos baseados no *middleware* Apache Cordova.

Ao longo do desenvolvimento da aplicação foram-se testando os middlewares mais utilizados pela indústria, foi uma maneira de perceber *in-loco* as capacidades e limitações de cada um. Testaram-se o Phonegap, Intel XDK e o Telerik.

O Phonegap é o *middleware* mais utilizado na indústria, contudo a existente versão 3.3.0 revelava-se instável, e a aplicação *crashava* após alguns minutos de utilização sem motivo aparente, algo que não acontecia com os outros dois competidores. A escolha recaiu sobre o Intel XDK por oferecer a segunda solução mais madura.

3.3.5.3 – Intel XDK

O Intel XDK fornece um ambiente de desenvolvimento multiplataforma abrangente para a criação de aplicações HTML5 com CSS3 e Javascript. Utilizando o *middleware* Apache Cordova, o Intel XDK converte o conteúdo web para código nativo, realizando uma série de otimizações para os processadores ARM. O resultado é uma aplicação nativa, com uma Responsividade muito próxima de uma aplicação nativa pura. Possui uma IDE próprio que auxilia o processo de desenvolvimento, depuração e *profiling* de performance nas métricas típicas.

Este *software* possui ainda alguns extras interessantes baseados nos serviços *cloud* da AppMobi, que permitem publicar a aplicação em *cloud* ou nos mercados mais populares, explorar a vertente de monetização e promoção.

3.3.5.4 – Responsive Design

Passada a barreira da plataforma, surge um novo obstáculo, a diversidade de formato de ecrãs inter e intra plataformas. Uma resposta ao problema é o “Responsive Design”, uma abordagem que esquematiza o design em grelhas expressas em unidades percentuais, que se adaptam ao tamanho do ecrã e em “*@media queries*” que atribuem regras específicas de CSS consoante a *querie*. Uma *@media querie* recebe como parâmetros os tamanhos do ecrã, proporções e orientação, desta forma é possível cobrir todas as combinações de formatos de ecrãs. Foram suportados os ecrãs do iPad, tablets Android de 10’ e 8’.

3.3.5.5 – Implementação

A estruturação da aplicação foi inspirada na *pattern* MVC (Figura 10). Esta *pattern* traz diversas vantagens no processo de desenvolvimento, tendo em conta que se está num desenvolvimento de um protótipo em que a sua evolução passa por um número sucessivo de maquetes. Torna-se necessário adotar uma estratégia que promova a reutilização de código nas diversas camadas, e que as alterações de cada uma não interfira nas demais. Outra vantagem deste modelo prende-se com a versatilidade no controlo de qualidade de cada *layer* com a possibilidade de avaliação com testes unitários.

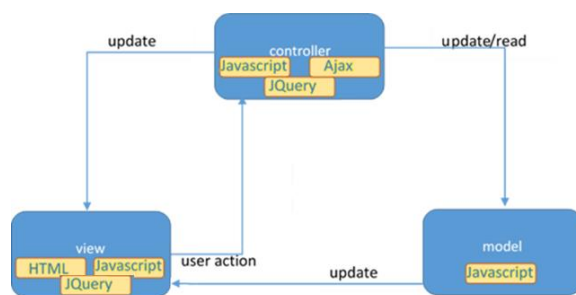


Figura 10

3.3.5.5.1 - Model Layer

Esta camada possui uma modelação dos dados que estão na base da aplicação, quer sejam dados de conteúdo, ou dados de configuração. Todos os objetos estão modelados em javascript.

Para a modelação dos dados de conteúdo (*People*, *News*, *Products*, *Definitions* e *Places*) usaram-se *patterns* de herança, pois cada um destes herda os atributos e funções de um objeto “*ContentObject*” genérico, sendo que cada um faz a sua implementação específica.

Sendo o Javascript uma linguagem interpretada e de tipagem dinâmica, em que tudo são *objetos* e não existe o conceito de *classe*, os mecanismos de herança diferem dos mecanismos de herança clássicos. Estes mecanismos são baseados em protótipos, uma propriedade comum a todos os objetos, que indica o objeto pai que lhe deu origem. Para simular uma relação de herança, este campo é preenchido com o objeto a que se deseje ser de hierarquia superior. [30]

E.g: `Person.prototype = new ContentObject();`

O objeto *ContentObject*: Objeto de conteúdo genérico, possui os atributos comuns de um objeto de conteúdo, *id* e *hash* e as funções “abstratas” `calcHash()` e `render()`.

Os objetos *Person*, *News*, *Definitions*, *Product*, *Place* e *Timeline*: Objetos hierarquicamente inferiores e que estendem diretamente a um objeto *ContentObject*. Fazem *override* aos métodos `calcHash()` e `render()`, no sentido que cada um é responsável por calcular o seu *hash* baseado nos seus próprios atributos, e por saber modificar a camada *View* com as suas próprias características via seletores

jQuery. Esta solução é eficaz, contudo induz algum acoplamento entre a camada View e a camada Model.

A camada *Model* possui também um objeto *Configuration* como uma representação das definições da aplicação. Neste grupo ainda se insere o objeto *Communication*, que encapsula todos os métodos usados na comunicação com o servidor. A comunicação é feita via pedidos Ajax assíncronos, são formatados em *Jsonp* para contornar a restrição das políticas de *cross-domain* dos pedidos *XMLHttpRequest* usados pelo Ajax. A política *cross-domain* impõe que um pedido só é efetuado com sucesso se o destino for coincidente com o servidor da página.

Nos anexos do capítulo 3 é possível consultar o diagrama de “classes” completo.

3.3.5.5.2 - Controller Layer

Esta camada faz a maioria da orquestração entre as camadas *View* e *Model*, engloba os objetos *GlobalOrchestrator*, *ViewOrchestrator* e *CommunicationHandler*.

O objeto *GlobalOrchestrator* é responsável por manter a aplicação num estado emparelhado com o servidor numa dada sessão *session_id*, e receber novos conteúdos em sincronização com a sessão. Quando novos objetos de conteúdo chegam, este notifica o *ViewOrchestrator*. Para este comportamento o objeto está permanentemente em estado de *short-polling* ao servidor em busca de informação.

Este objeto segue o seguinte diagrama de atividade:

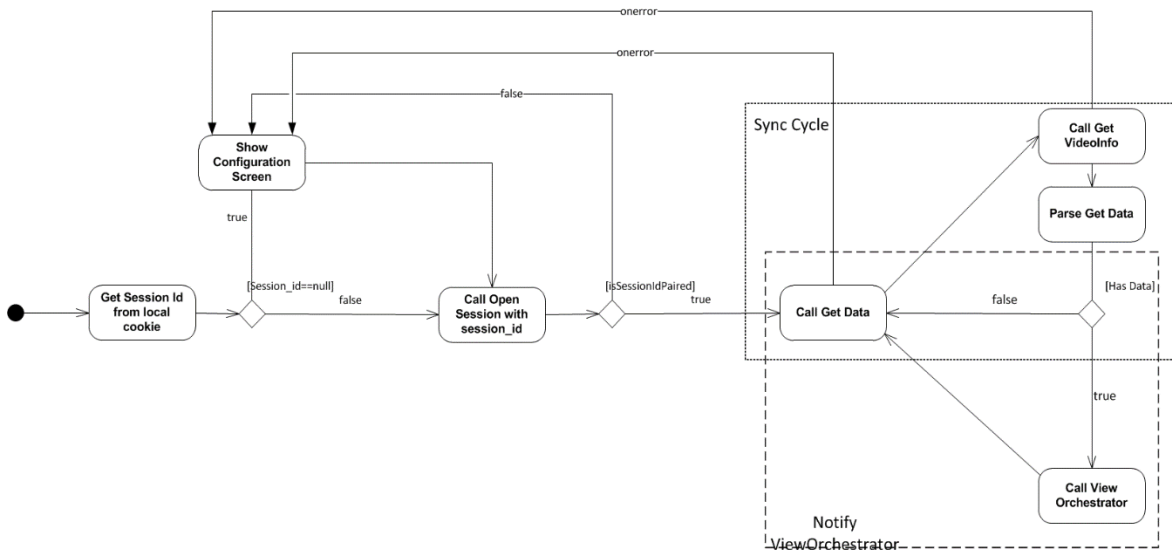


Figura 11:Diagrama de Actividade do Globalorchestrator

O objeto *ViewOrchestrator* é responsável pela gestão da camada *View* tendo em conta os estados da aplicação de modo “*stream*” (renderização do objeto mais atual na *timeline*) ou modo “*object*” (renderização de um objeto escolhido pelo utilizador). Quando o estado entra em modo “*object*”, é registado um *timeout* para regressar ao modo “*stream*”. Possui também todos os *listeners* JQuery à escuta de interações do utilizador na camada *View*.

O objeto *ViewOrchestrator* segue o seguinte diagrama de atividade:

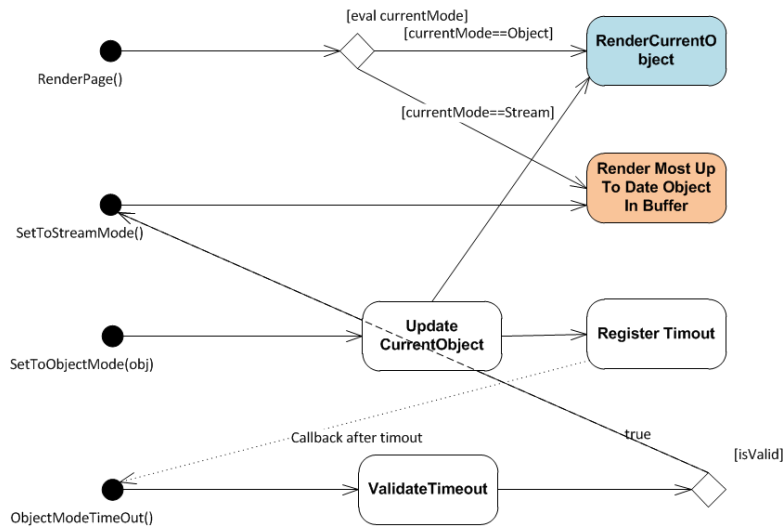


Figura 12: Diagrama de Atividade do ViewOrchestrator

3.3.5.5.3 - View Layer

Esta camada é responsável pela apresentação dos dados e pela captura da interação do utilizador, está implementada em *Html* com *Css3* e *JQuery*.

O modelo HTML é composto por uma grelha fixa, cujas células são atualizadas dinamicamente nos seus elementos da árvore DOM via *JQuery* pelas entidades das camadas *Controller* e *Model*.

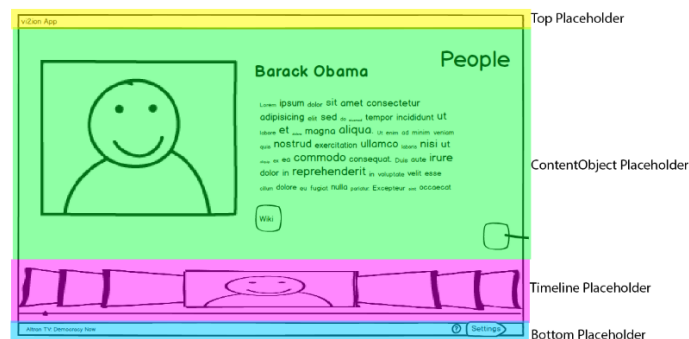


Figura 13: Organização do Html

A renderização e os mecanismos de captura dos eventos do utilizador da área “*ContentObject Placeholder*” ou “*Timeline Placeholder*” é efetuada pelo método *render()* específico de cada objeto da camada *Model*, ao que é associado um determinado *template* Html consoante o seu tipo.

3.4 Conclusão

A Maquete V2 foi desenvolvida com o intuito de solidificar e dar continuidade à prova de conceito da Maquete V1 como experiencia *second-screen*. Foi desenvolvida uma versão em contexto VOD que demonstrou as capacidades da tecnologia ACE do parceiro escolhido. A aplicação foi desenvolvida com a tecnologia híbrida do Intel XDK, tendo o seu desenvolvimento baseado em tecnologias web: *Html5*, *Css3* e *Javascript*.

O balanço foi bastante positivo, com a conclusão da Maquete V2 a Altran selecionou o vi2ion para exibir na Mobile World Conference 2014 em Barcelona, uma feira internacional de referência para o sector mobile, com uma afluência de 85000 pessoas. O vi2ion esteve em exposição durante 4 dias, apresentado pelo *Practice Manager* Engº Bruno Coelho. O feedback foi encorajador despoletando interesse nos colegas da Altran e na abertura de oportunidades de negócio do sector multimédia. O seguimento do conceito para a MaqueteV3 mostrava-se essencial.

4 - Maquete V3

Com a Maquete V2 foi demonstrado o conceito de ACE do parceiro Leankr, a espinha dorsal da aplicação vi2ion. A Maquete V3 pretende complementar este conceito com a inclusão de suporte a emissões *live*, e estender a aplicação com os módulos *Social* e *OTT* de forma a enriquecer a experiencia de utilização, e demonstrar de forma objetiva ao cliente as competências da Altran nestas áreas.

No capítulo II citou-se o estudo da sociedade *Second Screen Society* [1], em que foi analisada a tendência do mercado, os novos hábitos de consumo e em que medida as soluções existentes se afirmavam. Para classificar e diferenciar as diferentes propostas, este estudo identificou um conjunto de 5 categorias [2] em que uma aplicação *companion* idealmente deverá responder a um subconjunto para captar a atenção do mercado. Sumariamente as categorias são *controlo*, *conteúdo aumentado*, *social*, *discovery*, e *multe ecrã*. A Maquete V2 respondeu à categoria de *conteúdo aumentado*, enquanto a Maquete V3 pretende complementar o trabalho existente com a categoria *social* e mostrar capacidade para responder à categoria de *multe ecrã* caso o cliente posteriormente assim o deseje, através da inclusão de um *player vídeo OTT*.

Na sua conceção a MaqueteV3 tem como base um *refactoring* da MaqueteV2 segundo a *framework* Backbone.js, esta *framework* formaliza o modelo MVP, derivado do MVC, aumentado a organização e a qualidade final do código.

4.1 -Requisitos

Tabela 5: Requisitos MaqueteV3

Feature type	ShortName	Impacts Tablet	Impacts Server	Impacts TV	Type	Priority
Live Support	Support live channels	1	1		Functional	High
OTT (VOD) Support	OTT Vídeo - Low/High resolution support	1	1	1	Functional	High
	Backoffice to Manage OTT Vídeo Assets		1		Non Functional	Medium
Gamification	Gamming - Quiz	1	1		Functional	Medium
Social Interaction (Facebook)	Login	1	1		Functional	High
	Share	1	1		Functional	High
	Show Profile	1	1		Functional	High
	Proxy on Server Side		1		Functional	High
User Interface	New UI/UX	1			Functional	High

Feature type	ShortName	Impacts Tablet	Impacts Server	Impacts TV	Type	Priority
	Responsive Design with Grids	1			Non Functional	Medium
	Small Screens Support					Medium
V2-Code Refactoring	MVC - Backbone.js	1			Non Functional	High
Security	HTTPS				Non Functional	Medium

À semelhança da análise de requisitos da Maquete V2, foram igualmente estabelecidas prioridades aos requisitos. Nesta maquete as prioridades estão voltadas para a componente social e para a inclusão de um *player* OTT. Contudo estes requisitos prioritários dão origem a novos requisitos. Nomeadamente no *player* OTT, é conveniente ter um BackOffice simples de forma a organizar as fontes de vídeos. Na componente social, é necessário criar uma *proxy* do lado do servidor de forma a ultrapassar o obstáculo da comunicação de *Cross-Domain* entre a *App* e a API do Facebook. Em consequência das novas funcionalidades surge a necessidade de adequar o UI e a UX, pelo que este requisito também é denotado como alta prioridade, dentro deste grupo o suporte a pequenos ecrãs também está contemplado. Ainda foi manifestado o interesse em sincronização com canais *live*, devido às limitações atuais do parceiro, este suporte ainda só alcança alguns canais franceses. A partir desta etapa o desenvolvimento fica completamente endereçado ao estagiário.

Na conceção da MaqueteV2 o Phonegap encontrava-se na versão 3.3.0 e mostrava alguns sinais de instabilidade (a aplicação *crashava* apos alguns minutos sem motivo aparente), pelo que se optou pelo IntelXDK que se mostrava mais estável. Contudo a componente que fazia uso da *web-view* nativa dentro do contexto da aplicação (*inAppBrowser*) não permitia a customização dos controlos. Com a saída do Phonegap na versão 3.4.0, que nos testes se revelou mais estável, e de forma a colmatar as limitações do *inAppBrowser* do IntelXDK, passou-se a usar como middleware a última versão do Phonegap.

4.2 -Refactoring - Frameworks MV* (MVC, MVP, MVVM ...) [1]

Embora seja possível desenvolver uma *webapp* recorrendo puramente ao Javascript, atualmente esta prática é evitada através do uso de uma *Framework* Javascript MV*. Atualmente existem dezenas de *frameworks* que formalizam este conceito, fruto da maturidade das tecnologias web como Html5, Css3 e Javascript. Escolher uma *Framework* MV* é muitas vezes um desafio.

A família MV* descende do conceito original MVC. O MVC é um *design pattern* que encoraja uma certa organização estrutural do código, através da separação de conceitos. Reforça o isolamento da representação dos dados (*Models*) das interfaces (*Views*), com uma terceira componente a unir estas duas e controlar a lógica de negócio (*Controller*). Esta *Pattern* foi usada a primeira vez no final dos anos 70 na conceção do smalltalk-80.

Ao longo dos tempos até à computação moderna, este padrão foi sendo aplicado a um grande conjunto de linguagens, incluído o Javascript, o foco deste estudo. O Javascript pegou no conceito

original MVC e foi adaptando-o às suas necessidades, dando origem a varias variações que estendem o conceito original, MVC, MVP e MVVM são as mais comuns.

Para o conceito original MVC tem-se as seguintes abstrações:

Models: Gerem unicamente os dados da aplicação. Quando um modelo sofre uma alteração de estado (*update*) informa os seus *observers*, e estes invocam um procedimento específico.

Views: São as representações gráficas dos *Models* e do seu estado. Enquanto no Smalltalk as *Views* eram responsáveis para renderizar um bitmap, no Javascript servem para construir e manter um elemento DOM. Tipicamente uma *View* observa um *Model*, e é notificada para atualização quando este muda de estado.

Controllers: São um intermediário entre as camadas *Models* e *Views*, e estão classicamente responsáveis para atualizarem os *Models* e as *Views* quando o Utilizador manipula a Interface *View*, consoante as regras de negócio.

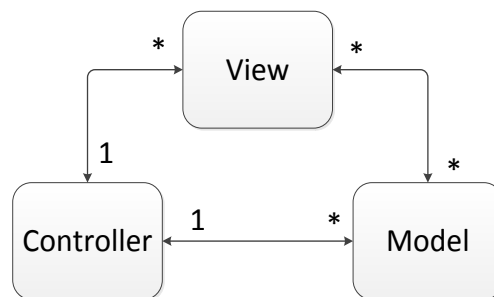


Figura 14:MVC Clássico

Alguns exemplos de *frameworks* que são estritamente fiéis ao modelo são: Maria.js, PureMVC e CanJs.

4.2.1 - MVP

Surgiu no início dos anos 90 por via da empresa Taligent, tendo sido aproveitado nas recentes arquiteturas web. É um derivado do MVC, com foco na camada de apresentação. Apresenta algumas diferenças fundamentais ao MVC. A primeira diferença visível na denominação é o P de *Presenter*, que tal como o *Controller* é a componente que tem a interface das regras de negocio para a *View*. Contudo ao contrário do MVC, as invocações das *Views* são delegadas apenas ao *Presenter*, e as alterações do estado do *Model* apenas são observadas pelo *Presenter*. A relação de cardinalidade de *Presenter/View* é de 1:1.

Solicitados por uma *View*, o *Presenter* executa o trabalho relativo ao pedido do utilizador, e o resultado é devolvido para a *View*. Os *Models* podem despertar eventos na sua mudança de estado, mas apenas o *Presenter* tem o papel de os subscrever.

Esta arquitetura reduz ainda mais o acoplamento existente no modelo MVC, pois reduz as ligações entre camadas, aumentando a sua separação. Isto traz vantagens, aumenta a testabilidade do

sistema, pois cada componente tem menos ligações às restantes, e o comportamento de cada componente é mais facilmente reproduzido estaticamente (*mock*).

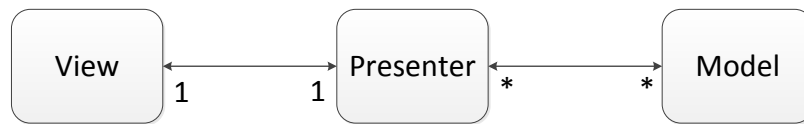


Figura 15:MVP

ex: BackboneJs, RiotJs, (Não Js: ASP.net, GWT, Swing)

4.2.2 - MVVM

Foi anunciado em 2005 pela Microsoft, concebido para o Windows Presentation Foundation (WPF) e SilverLight. Este *pattern* pretende conjugar as vantagens inerentes à separação de responsabilidades e baixo acoplamento do modelo MVP com as vantagens do conceito de *data-binding*, aplicado entre a *View* e a *ViewModel*. O *data-binding* mantém a *View* em permanente sincronismo com a camada *ViewModel*. Remove praticamente todo o código da camada *view* (GUI), deixando esta camada apenas com os elementos de design, sendo que o código da GUI é gerido pela componente *ViewModel*. Desta forma um *developer* de UI/UX foca-se somente na componente *View* e nas suas necessidades, deixando os *bindings* para a camada *ViewModel*. E a responsabilidade da lógica da UI são delegadas para os desenvolvedores da aplicação.

As responsabilidades de cada camada são as seguintes:

Model: Como em todas as arquiteturas da família MV*, a camada *Model* destina-se apenas em representar os dados de um domínio específico.

View: É a camada de aplicação destinada à interação com o utilizador, possui todos os elementos gráficos e componentes de interação. Ao contrário dos modelos MVC e MVP, esta camada não possui qualquer inteligência, é a representação gráfica em tempo real da camada *ViewModel*.

ViewModel: É o 'Modelo' em tempo real da camada *View* onde possui um *data-binding* direto. Possui igualmente a lógica de negócio, que se encontra intrinsecamente associada aos *bindings*. Observa de igual modo as alterações da camada *Model*, e é responsável pela sua atualização. A relação de cardinalidade para com a camada *View* é de 1:N.



Figura 16:MVVM

É um modelo bastante adotado na oferta de *frameworks* Javascript MV*, encontra-se presente no Angular.js, KnockoutJS, EmberJs, BackboneJs + MarionetteJs

4.2.3 - Critérios de Escolha Frameworks MV* [2][3]

Fruto da maturidade das tecnologias web, atualmente existem dezenas de *frameworks* que formalizam os conceitos da família MV*, cada uma com a sua interpretação. Escolher uma *framework* é muitas vezes um desafio.

O projeto TodoMVC [4] fornece um catálogo atualizado com as *frameworks* mais populares, e uma implementação em cada *framework* de uma mesma aplicação base. Permite uma fácil avaliação das potencialidades e limitações de cada *framework*. Para selecionar uma *framework* obviamente não basta comparar as implementações do projeto TodoMVC, Addy Osmany da Google sugere alguns critérios de seleção, onde serão aplicados sobre as *frameworks* mais populares de cada modelo MVC (CanJs), MVP (Backbone) e MVVM (Angular, Ember).

Quais as capacidades da framework? (features)

Esta questão prende-se com o nível técnico, que ferramentas a *framework* oferece e de que maneira permite endereçar os problemas. Tipicamente nas arquiteturas MV* valoriza-se o *data-binding* bidirecional, se possui suporte a algum motor de *templates* HTML para uma maior produtividade/flexibilidade, se possui ferramentas para sincronizar a camada *Model* com o servidor, ou se possui um motor de *routing* para guardar o estado entre vistas diferentes uma vez que uma web *app* é composta por uma única página.

Tabela 6: Features das Frameworks Javascript

	TwoWay-Data-Bindig	HTML-Template Engine	Model Sync	Routing
Angular	Built-In	Declarative DOM	None/Manual	Built-In
Ember	Built-In	Handlebar.js	Built-In	Built-In
Backbone	None/Manual	Supports any engine, bundled with underscore.js	Built-In (overridable)	Built-In
CanJs	Built-In	EJS4	Built-in	Built-In

A framework já deu provas em produção? Qual o grau de maturidade?

O melhor teste é o mundo real, e os seus desafios. O portfólio é um ótimo indicador que uma *framework* resulta em produção, e responde aos problemas inerentes a uma aplicação de larga escala. Propostas de *frameworks* muito recentes tendem a sofrer muitas atualizações e ter uma API instável. Para não ficar com o projeto em risco, e atravessar grandes períodos de *refactoring* convém investir numa *framework* que mostre sinais de maturidade.

Desta seleção todas deram provas em produção da sua tecnologia. Segue a tabela com os projetos mais relevantes:

Tabela 7:Portofólio das Frameworks Javascript

Large Scale Apps Portfolio	
Angular	Ps3 Youtube, The Guardian, Nike
Ember	Eloqua, Code School
Backbone	LinkedIn, SoundCloud, FourSquare
CanJs	Wallmart, store.apple.com

Possui uma boa base de documentação? Qual o tamanho da comunidade?

A documentação e ou o apoio da comunidade é fundamental para cada processo de desenvolvimento. Esta questão está quase sempre dependente do grau de maturidade da *framework*. Um bom indicador para a expressão da comunidade é o número de seguidores no github, o portal de eleição para estes projetos.

Todas as *frameworks* selecionadas disponibilizam uma documentação completa das suas api's, assim como exemplos básicos. Para estimar o tamanho de cada comunidade o Github apresenta o seguinte número de seguidores:

Angular.Js:26'000, Backbone.Js:18'000, Ember.Js:10'000, CanJs:960

Observamos que as mais estabelecidas são a Angular e o Backbone, porem também são as mais antigas. A Ember e a CanJs são mais recentes e por isso a comunidade ainda não expressa os valores das comunidades mais antigas, mas são as que têm registado a maior taxa de crescimento.

É flexível ou dogmática?

Cada *framework* segue a sua interpretação do modelo MV*, e cada uma oferece um certo grau de liberdade no modelo que apresenta. Com uma *framework* dogmática o programador tem menos espaço para as escolhas que faz, e a *framework* realiza grande parte do trabalho autonomamente. Traz a vantagem de uma maior produtividade, e de ser menos propício a más práticas de desenvolvimento. Em contrapartida apresentam uma maior curva de aprendizagem, cada uma com as suas próprias diretivas e jargões.

Do outro lado com uma *framework* flexível o programador tem uma grande liberdade, e é responsável por um grande número de decisões, cada componente permite uma maior customização. É comum apelidarem estas *frameworks* de livrarias. A curva de aprendizagem é geralmente mais baixa.

Backbone.JS: É a *framework* mais flexível deste conjunto, deixando o desenvolvedor com uma grande liberdade de decisões na arquitetura. Oferece uma solução minimalista e modular para a separação de responsabilidades, fácil de integrar com projetos já existentes.

Angular.Js: É uma *framework* bastante dogmática, com diretivas rígidas inspiradas no conceito CRUD das bases de dados relacionais.

Ember.Js: Com um grande compromisso com a produtividade, deste conjunto é a *framework* mais rígida, e que por isso tem a maior curva de aprendizagem.

CanJs: É uma *framework* bastante flexível ao estilo do Backbone.Js, acrescenta a funcionalidade de *data-binding* bidirecional, contudo apenas aceita o Mustache.Js como motor de *templates*.

Qual o tamanho da *framework* e respectivas dependências?

Em alguns requerimentos a pegada de memória pode ser um fator de escolha, pois só não afeta o tempo de carregamento, como pode afetar a performance de uso. Nesses casos convém não só analisar o tamanho da *framework*, mas também a soma das suas dependências.

	Sum Of Size (Kb)
Angular	33
Ember	78.8
Backbone	44.8
<i>underscore</i>	5.4
<i>backbone</i>	6.8
<i>jquery</i>	32.6
CanJs	47.2
<i>canjs</i>	14.6
<i>jquery</i>	32.6

4.2.4 - Escolha efetuada: Backbone.js

Observa-se que a nível de *features* as *frameworks* mais completas são a Angular.Js e a Ember.Js. Contudo muitas destas *features* transcendem os requisitos da aplicação da MaqueteV3, como por exemplo o *data-binding* bidirecional e o *routing*. São ainda *frameworks* bastante dogmáticas que impõem e direcionam o desenvolvimento segundo as suas diretrizes. Este aspeto não é de todo negativo, é uma forma da *framework* dar resposta a problemas comuns e assim aumentar a produtividade, mas traria alguns custos ao nível da integração com o já existente módulo de controlo, e uma maior curva de aprendizagem inicial com os seus dialetos.

Devido às restrições de tempo a MaqueteV3 aproveitou alguns módulos da camada de controlo da MaqueteV2, pelo que se valorizou o fator da flexibilidade das *frameworks* analisadas.

A escolha recaiu sobre a *framework* Backbone.Js. Possui mecanismos de comunicação entre camadas orientada a eventos, é uma *framework* estável com provas dadas em produção, tem uma boa base de documentação e uma extensa comunidade. A sua API permite reestruturar o código em modelos (Backbone.Models), coleções de modelos (Backbone.Collections) e *presenters* (Backbone.views) associadas a *templates* DOM. Dentro dos modelos analisados, onde mais se enquadra é no MVP, mas como nesta *framework* não existe nenhuma entidade dedicada ao controlo da lógica de negócio, muitos apenas enquadram o Backbone como sendo simplesmente MV*. A nível de arquitetura o Backbone no modelo MVP induz um baixo acoplamento entre camadas, o que numa fase posterior irá facilitar a condução dos testes unitários.

4.3 Arquitetura

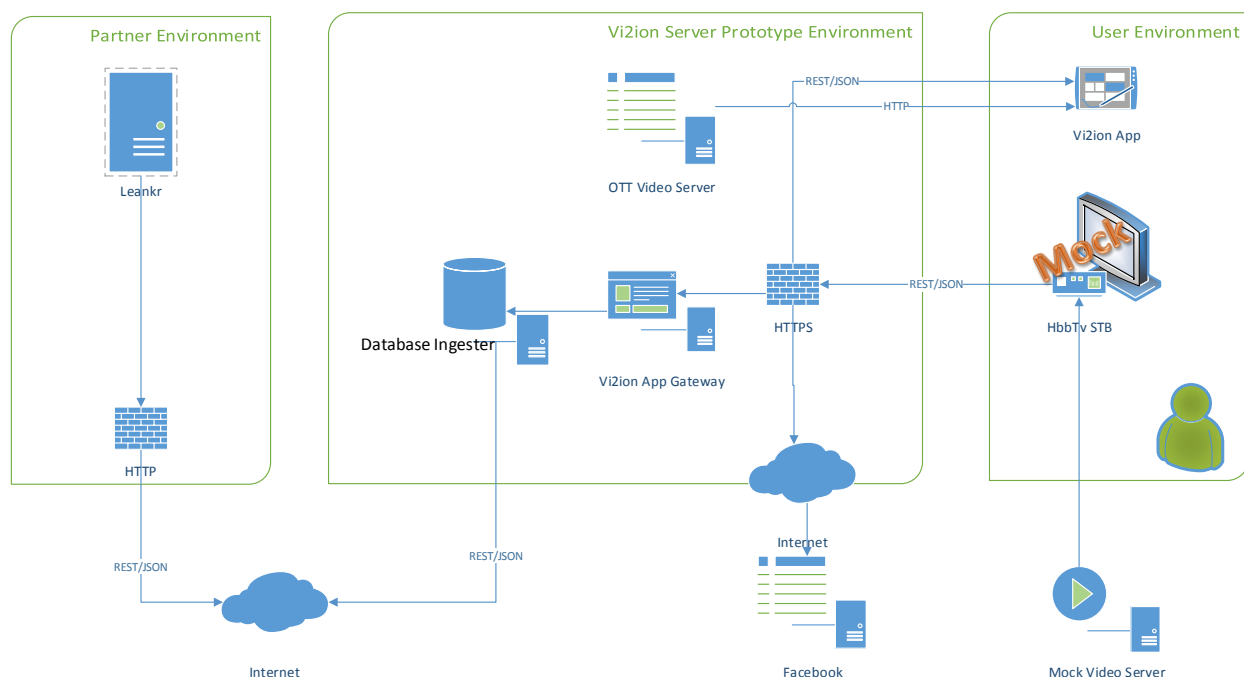


Figura 17 :Arquitetura Alto Nivel da MaqueteV3

Com os novos requisitos a paisagem da arquitetura de alto nível do sistema existente na MaqueteV2 foi complementada. As funcionalidades de suporte a canais em direto, o módulo social, e o módulo de vídeo OTT são suportadas pelo *Vi2ion Server Environment*.

4.3.1 - Vi2ion Partner Environment

Representa o serviço de ACE da Leankr. Este serviço fornece a base de suporte de enriquecimento de conteúdo para as emissões em direto da TV Francesa. Este serviço possui uma interface REST com estruturação em JSONP.

Métodos principais da Leankr API:

Tabela 8: Api do Serviço de ACE da Leankr

Método	Descrição
/people	Tendo como dados de entrada o identificador do canal e o instante atual, devolve um Array de objetos "Pessoa" que foram identificados para o dado instante. Cada objeto contem informações biográficas, fotos, noticias relacionadas e quizzes (quando aplicável), entre outros.

Método	Descrição
/news	Tendo como dados de entrada o identificador do canal e o instante atual, devolve um Array de objetos “News” que foram identificados para o dado instante. Cada objeto contém a <i>headline</i> da notícia, a imagem associada, e um link externo para a fonte.
/places	Tendo como dados de entrada o identificador do canal e o instante atual, devolve um Array de objetos “Lugares” que foram identificados para o dado instante. Cada objeto contém uma descrição, o <i>url</i> da API Google maps e fotos.
/products	Tendo como dados de entrada o identificador do canal e o instante atual, devolve um Array de objetos “Produtos” que foram identificados para o dado instante. Cada objeto contém o título do produto, uma descrição, e um link externo de comércio eletrônico.
/definitions	Tendo como dados de entrada o identificador do canal e o instante atual, devolve um Array de objetos “Definições” que foram identificados para o dado instante. Cada objeto contém o título da definição, uma descrição, e um link externo para a referência da definição.
/mashup	Agregação dos métodos <i>/people</i> , <i>/product</i> , <i>/news</i> , <i>/places</i> e <i>/definitions</i> numa única chamada.

4.3.2 - Vi2ion Server Prototype Environment

Foi adicionada a entidade *Database Ingestor*, como uma componente auxiliar para o suporte de canais em direto. Esta entidade comunica diretamente com os servidores da Leankr para a obtenção das referências para todos os canais. As referências são armazenadas em base de dados, para posteriormente serem acedidas pela entidade *Vi2ion App gateway*. Funciona como uma cache da Leankr, otimizando assim o tempo de resposta do sistema, e diminui consideravelmente o número de pedidos à Leankr, cuja API tem o *throughput* limitado.

Como consequência destas alterações, a api do servidor foi adaptada para o efeito:

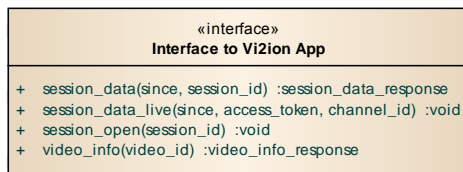


Figura 18: Interface do Servidor para a App

Foi adicionado o método *session_data_live* que retorna os dados de enriquecimento de conteúdo sincronizados com o canal de tv *channel_id*, desde o instante *since*. O campo *since* representa o *offset* da *stream* em que foram coletados os últimos dados, se este valor for 0, retorna todos os dados desde o início da *stream*. Este campo é representado no formato Unix Timestamp. O campo *access_token* é utilizado para verificar as permissões do pedido.

Os módulos OTT e Social serão analisados em detalhe posteriormente.

4.3.3 - Base de Dados

A base de dados teve por base a configuração existente na MaqueteV2 e ao seu contexto VOD. Para as novas funcionalidade de suporte emissões em direto, social e vídeo OTT foram adicionadas novas entidades. O diagrama atual de entidades e relacionamentos é o seguinte:

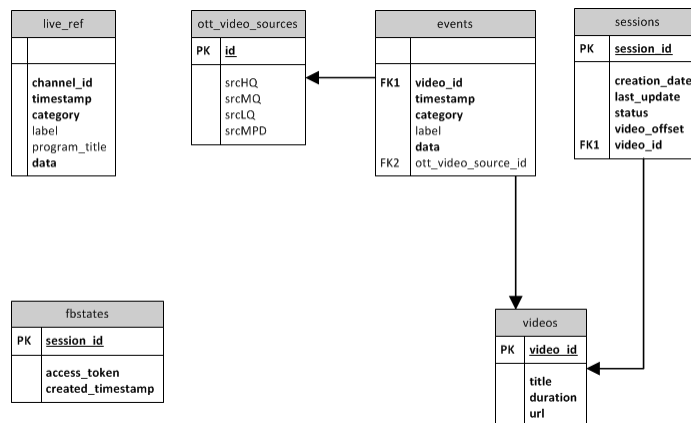


Figura 19: Diagrama de Entidades da Base de Dados da MaqueteV3

Em relação à configuração da base de dados da Maquete V2, mostram-se as seguintes novas entidades:

Live_ref: Representa as referências para emissões *live* de canais franceses, possuem a mesma estrutura das referências VOD da tabela **events**, com a inclusão do campo *program_title*. Os dados desta tabela chegam diretamente da API da Leagr e são inseridas pela entidade Database Ingester.

Fbstates: Destina-se a ser usada por um serviço de Proxy que faz a mediação entre a aplicação móvel e a API do Facebook. Guarda o estado e os *tokens* de acesso para cada sessão aberta.

Ott_vídeo_sources: Representa as fontes para um conteúdo OTT, para o mesmo conteúdo contem uma fonte de *stream* adaptativo *srcMPD*, e fontes de *stream* vídeo normal (mp4,webm,ogg) *srcHQ*, *srcMQ*, *srcLQ* para alta, media e baixa qualidade respetivamente.

Events: Para o suporte de vídeo OTT, cada referencia VOD poderá ser associada a um conteúdo de vídeo OTT *content_video_source_id* da tabela **ott_vídeo_sources**.

4.3.4 - User Environment - Aplicação Segundo o Modelo MVP (Backbone.js)

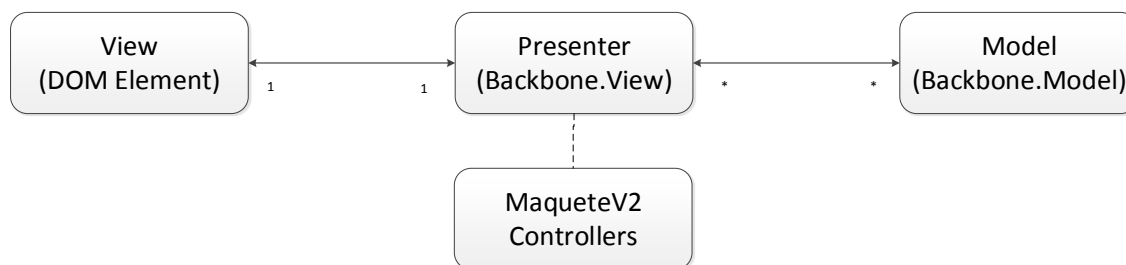


Figura 20: Modelo MVP utilizado na App

A *framework* Backbone sugere uma separação de responsabilidades segundo o modelo MVP, onde para a camada *Model* é implementada pelo objeto Backbone.Model, a camada *View* implementada pelo motor de *templates* DOM, e pela camada *Presenter* responsável pela lógica da sua *View* correspondente, esta camada é implementada pelo objeto Backbone.View. A Backbone não prevê uma camada de controlo, deixando essa responsabilidade ao critério do programador, para este efeito foram reutilizados os módulos de controlo da MaqueteV2 (*GlobalOrchestrator.js*, *ViewOrchestrator.js*, *CommunicationHandler*), foi ainda adicionado a componente de controlo *FacebookOrchestrator* para a funcionalidade social. Por esta camada não se inserir na definição formal do modelo MVP, também é aceitável classificar a arquitetura como MV*. Segue uma sumária explicação de cada camada, o diagrama completo de Entidades e Objetos encontra-se em anexo.

4.3.5 - Camada Model:

A base de modelos da aplicação é composta pelos objetos *PeopleModel*, *PlaceModel*, *ProductModel*, *NewsModel* e *DefinitionsModel*. Estes objetos representam os dados dos conteúdos a serem apresentados na aplicação, todos estendem o objeto *ContentModel*, que por sua vez estende o objeto Backbone.Model. O objeto *ModelCollection* é responsável por manter uma coleção finita destes modelos *ContentModel*, ordenados temporalmente em que são evitados duplicados. A cada alteração de estado, este objeto notifica o bus de eventos que houve uma alteração.

O objeto *LivechannelModel* representa a informação de um canal, com o seu id de sincronização, o objeto *LivechannelModelCollection* representa a respetiva coleção, com métodos para sincronizar esta lista com o servidor.

Acrescenta-se ainda o objeto *ConfigurationModel*, que é responsável pelo armazenamento de definições de *pairing*, controlo e visualização.

4.3.6 - Camada Presenter:

A camada Presenter é responsável pela lógica e eventos da UI da camada View. É representada pelos objetos *PeopleView*, *PlaceView*, *ProductView*, *NewsView* e *DefinitionsView*. Estes objetos descendem do objeto *MainContentView* que contem os métodos principais, que são *overriden* consoante o tipo. O objeto base tem a seguinte conceção:

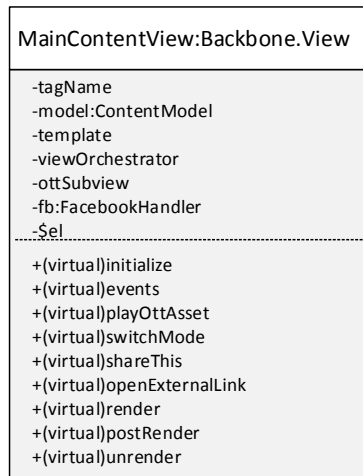


Figura 21:Backbone Presenter MainContentView

Os objetos principais deste objeto são o *model*, que é a referência para o Modelo específico. O objeto *template*, que é a referência para o seu *template* específico HTML. O objeto *fb* que é uma referência ao objeto *FacebookOrchestrator* para as funcionalidades do Facebook. Para a reprodução de vídeos OTT existe o *Presenter OTTView* dedicado à sua renderização, este contém os mecanismos necessários, nomeadamente a referência ao *player* Mpeg dash, este *Presenter* irá ser detalhado no capítulo “OTT”. Por fim o objeto *\$el* que representa a referência para o seu elemento da camada *View* (elemento DOM), cada *MainContentView* contém um elemento DOM *\$el*.

Existe também o *Presenter CollectionView*, responsável pela gestão do objeto *timeline da camada View*. Tem a função de apresentar as referências ordenadas temporalmente, implementar a lógica do *slider*, e alterar o seu estado consoante a interação do utilizador, ou pela chegada de novas referências. O *Presenter LiveChannelListView*, responsável por apresentar um pequeno comando no ecrã principal, que permite comutar o canal a ser apresentado pela aplicação em modo *live*. Finalmente o *Presenter ConfigurationView*, responsável pela *View* do menu de configurações, em que o utilizador define os dados de sincronização, selecionar o canal, selecionar o tamanho do *buffer*, entre outras definições.

4.3.7 - Camada View:

Esta camada possui unicamente os elementos de apresentação, e a capacidade de direcionar os eventos despertados pelo utilizador para o seu *MainContentView* da camada *Presenter*. Corresponde a um elemento HTML existente no DOM, este elemento foi formatado a partir de um *template* específico da livreria *Underscore.js*. Este elemento está ligado a um e um só *Presenter*.

Existem *templates* para cada tipo de referência (*People,Products,News,Places,Definitions*), para a *timeline*, para o menu de configurações, para a janela de informações e para o comando da seleção de canais. Estes encontram-se embebidos no *index.html*, contudo o motor de *templates Underscore.js* permite que sejam carregados dinamicamente de outra fonte.

Todas as componentes gráficas de UI foram aproveitadas da biblioteca JQuery Mobile [31], nomeadamente o tema da página [32], os mecanismos de *input* do menu de configurações, os *modal dialogs* para o *share* social, mas sobretudo as *responsive-grids*. As *responsive-grids* do JQuery Mobile foram um elemento fundamental para o desenvolvimento da aplicação, e para o seu suporte aos variados ecrãs. A estrutura gráfica foi toda concebida neste formato, onde é possível que alguns dos seus elementos internos se ajustem dinamicamente ao ecrã. Facilitando assim o desenvolvimento da UI para diversos dispositivos. A organização do *layout* segue a seguinte estrutura de *grids/cells*:

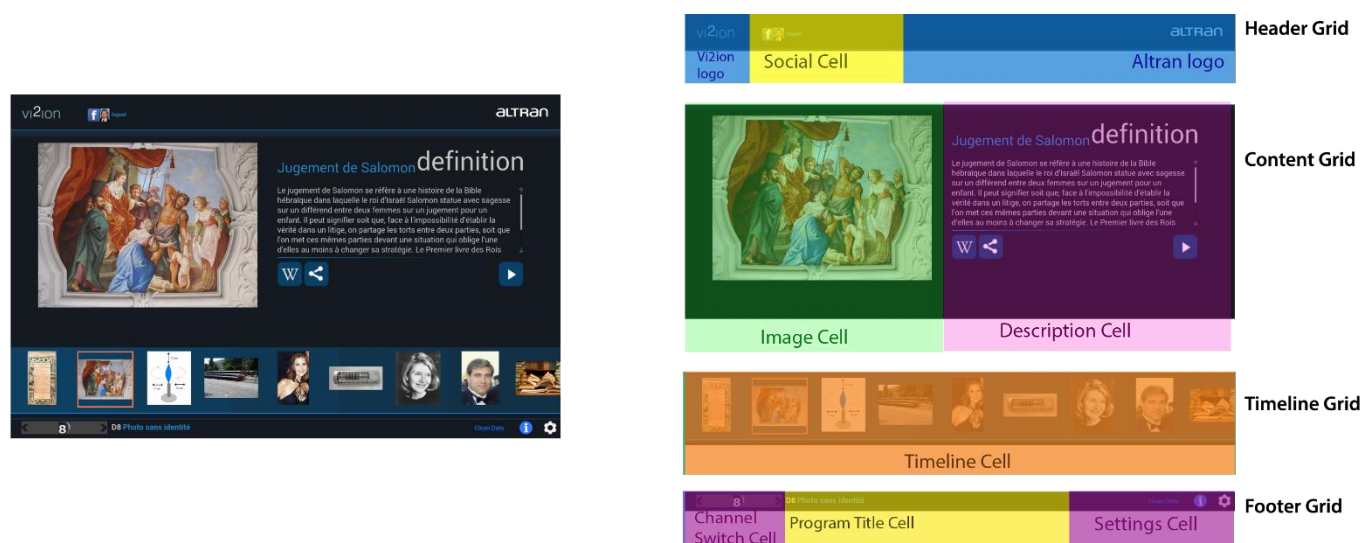


Figura 22: JQuery Mobile Responsive Grids

Começa por uma zona de cabeçalho, onde se encontram o *branding* da aplicação e uma zona de *login/logout* da rede social Facebook, caso se encontre num estado de *login*, a foto de perfil é mostrada. De seguida existe a *grid* do conteúdo, é aqui onde serão mostradas as referências (*people*, *products*, *news*, *places* e *definitions*) em conformidade com os seus respetivos *templates*. De seguida existe a *timeline* das referências, onde é possível comutar a referência ativa. E finalmente existe o rodapé, onde é possível trocar o canal ativo (se estiver em modo *live*), ver o título do programa, e aceder às definições. As *grids* ajustam-se ao tamanho do ecrã, respeitando as percentagens que lhes foram definidas, contudo essas percentagens necessitam de ser alteradas consoante o *aspect-ratio* do ecrã. Por este motivo as percentagens são reajustadas por *Css3 media queries* consoante o tipo de ecrã. Os ecrãs suportados são iPhone, iPad, Android 10', Android 8' e Android 5'.

Exemplo da adaptação do *Responsive Design* aos diferentes ecrãs:

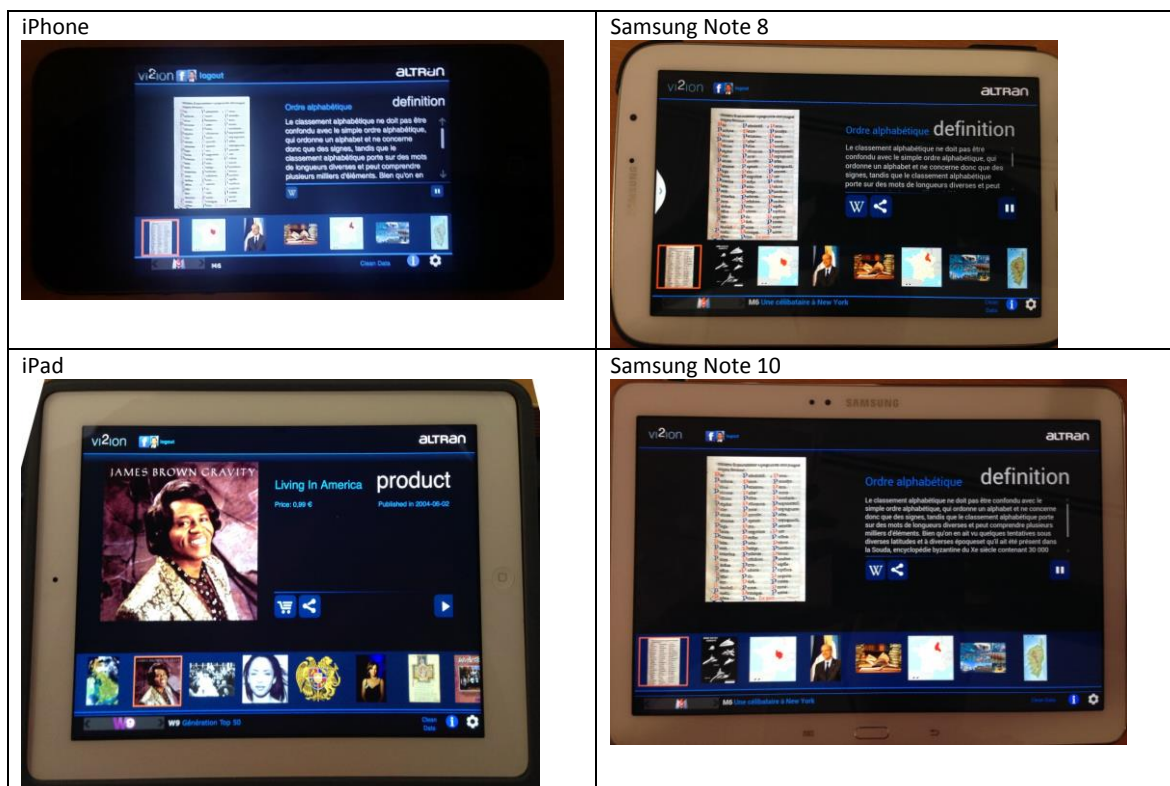


Figura 23: Responsive Design, Multi-Ecrãs

4.4 - Social (Facebook)

O Social tornou-se um requisito universal em qualquer aplicação móvel ou web, o vi2ion naturalmente seguiu esta tendência. Permite ao utilizador publicar conteúdos do vi2ion no seu mural, relacionados com o conteúdo que está a ser mostrado com a aplicação. A rede social escolhida foi o Facebook, esta foi escolhida devido à sua popularidade em todo o mundo, e em especial na Europa, onde estão os maiores clientes da Altran do sector multimédia.

O Facebook oferece SDK's nas plataformas mais populares para facilitar a integração com aplicações. Oferece SDK's para aplicações nativas em iOS, Android e Windows Phone, e para páginas ou aplicações web através da SDK Javascript^[5].

A Api do Facebook implementa o protocolo de autenticação HTTP OAuth 2.0, á semelhança de outras empresas como twitter, hotmail, paypal etc. O principal objetivo do Oauth é permitir um terceiro (*third-party*) aceder a recursos protegidos de um *Resource Server* que são propriedade do utilizador final (*Resource Owner*), sem que este chegue a partilhar as suas credenciais com o terceiro (*third-party*). Este acesso é feito de forma controlada e volátil. Mais detalhes sobre o protocolo podem ser encontrados na respetiva especificação IETF ^[6].

4.4.1 - Obstáculo das aplicações híbridas com o OAuth:

O OAuth prevê 2 perfis de utilização^[6]: *User-agent-based-application*, é um cliente público que corre num *browser* ou *web-view*, em que o seu código foi descarregado de um web-server, e é executado num dispositivo controlado pelo dono dos recursos alvo. Neste perfil a entidade responsável por ceder os acessos aos recursos (*Authorization Server*) necessita saber qual é o *web server* autorizado, e apenas aceita pedidos vindos deste; O segundo perfil é o *Native application*: É um cliente público, instalado e executado num dispositivo usado pelo dono dos recursos alvo, é assumido que nenhum dado de autenticação da aplicação pode ser extraído desta. O Facebook cobre estes dois perfis distinguindo-os como sendo aplicação web ou nativa, para aplicações web oferece o SDK Javascript, e para as aplicações nativas oferece SDK's para iOS, Android e WindowsPhone.

O problema das aplicações híbridas é que não se enquadram em nenhum destes dois perfis, uma aplicação híbrida é implementada com tecnologias web, mas é executada no dispositivo móvel como sendo uma aplicação nativa. Se for usada a Facebook Javascript SDK numa aplicação híbrida, os pedidos são barrados como sendo pedidos *cross-domain*, uma politica de segurança dos pedidos XMLHttpRequest usados pelo Javascript, que bloqueia pedidos feitos a *hosts* diferentes do *host* da página, que no caso das aplicações híbridas o host é o próprio dispositivo^[7]. Ainda que esta barreira fosse ultrapassada, o *Authorization Server* nega todos os pedidos que não venham do *host* que aloja a aplicação (definido no *backoffice*), ora sendo o *host* o próprio dispositivo, todos os pedidos irão ser barrados.

Com todas estas limitações, não existe nenhuma SDK oficial que dê suporte a aplicações híbridas. Existem duas maneiras de contornar este obstáculo:

1 - Construir uma *proxy* alocada num *host* acessível publicamente, em que a proxy serve de intermediário entre os pedidos da aplicação e a Graph API do Facebook. Desta forma é possível registar a aplicação no Facebook como sendo uma aplicação web, correspondente ao perfil *User-agent-based-application* do OAuth. O Facebook sugere os passos para a construção “manual” do processo de login na sua página de desenvolvimento ^[8].

2 – Usar *plug-ins* nativos do Phonegap, uma vez que é a plataforma usada e esta permite a instalação de *plug-ins* nativos e usá-los através de uma interface Javascript. É uma ponte entre código nativo da plataforma, e código web. Desta forma é possível registar a aplicação no Facebook como sendo uma aplicação nativa, correspondente ao perfil *Native application* do OAuth.

A escolha efetuada foi a Proxy, uma vez que resulta num esforço de implementação significativamente menor, é uma solução que possui uma interface REST acessível independentemente da plataforma utilizada.

4.4.2 - Arquitetura da Vi2ion FB Proxy

O primeiro passo para criar uma aplicação dentro do Facebook para um serviço já existente é criar uma *app* no portal *developer.Facebook.com*. Este portal oferece diversas funcionalidades e ferramentas, contudo o foco irá incidir sobre os detalhes da aplicação e na definição do processo de autenticação. Neste caso o serviço vai ser representado pelo vi2ion FB Proxy, sendo que esta é a única entidade permitida a comunicar com a *api* do Facebook em nome da aplicação vi2ion. O tipo de aplicação registado é como sendo uma aplicação web, não nativa. Após este registo é fornecido um *app_id* e um *app_secret* para a aplicação vi2ion, estes *tokens* destinam-se ao processo de autenticação, e são trocados unicamente na comunicação entre a *proxy* e a *api* do Facebook. Todas as comunicações que envolvem a *proxy* são encriptadas pelo protocolo Https, .

A *proxy* foi criada como um componente extra no servidor vi2ion *app gateway*, e é servida por uma interface REST. Foram implementadas as funcionalidades de *login*, *logout*, *share* e detalhes pessoais da conta.



Figura 24: Interface da Facebook Proxy

A lógica do lado da aplicação móvel/cliente, estas funcionalidades estão implementadas pelo objeto de controlo *FacebookHandler*. Seguem-se os diagramas de sequência para cada método da API da Proxy:

4.4.2.1 - login:

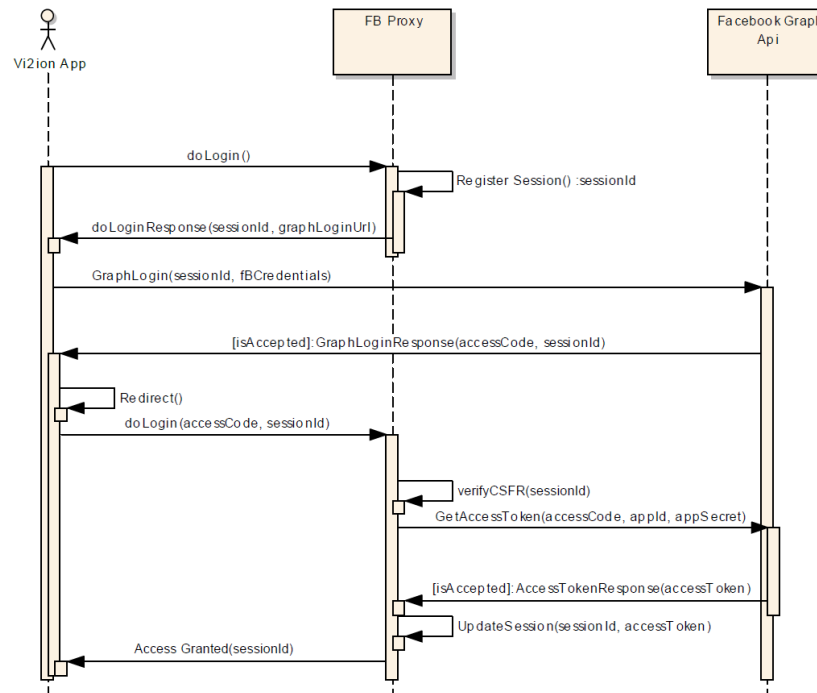


Figura 25: Sequencia do Login

Nesta sequência o primeiro passo do cliente é pedir o *Url* de login no Facebook à *proxy*. A *proxy* guarda a referência da sessão em persistência, e responde com o *Url* de login *graphLoginUrl*. O login de *url* é composto pelo *url* do serviço de login do Facebook, e ainda com os parâmetros *redirect_uri* e *sessionId*. O ecrã de login é aberto numa nova web-view nativa, dentro do contexto da aplicação. Após o login com sucesso do lado da Api do Facebook, esta retorna ao cliente uma página em branco, com *header* em *redirect* para um novo *url*, composto pelo *redirect_uri* e pelos parâmetros *accessCode* e *sessionId*, significando que o acesso foi concedido.

O *accessCode* é redirecionado à *proxy* e associado em persistência ao *sessionId*. O *accessCode* tem um carácter volátil, apenas serve para pedir o *accessToken*, pelo que só pode ser utilizado uma única vez. O *accessToken* tem um carácter mais persistente, possui uma validade de algumas horas. Para o obter, a *proxy* invoca o pedido à Api do Facebook com os parâmetros *accessCode*, *appId* e *appSecret*. Se este pedido for validado, a Api do Facebook retorna à *proxy* o *accessToken*, esta atualiza o estado e informa o utilizador do sucesso do login. Caso o *accessToken* se verifique expirado, o processo de login terá de ser reiniciado.

Exemplo da *web-view* nativa do ecrã de login, e respetiva confirmação após o sucesso:

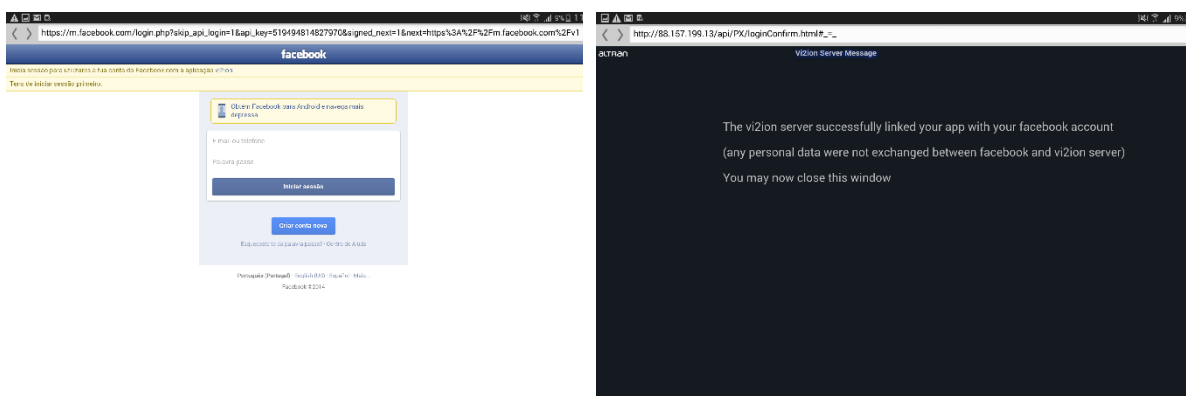


Figura 26: Ecrã de Login e Confirmação

O método *verifyCSFR* é um procedimento de segurança executado em todos os pedidos feitos à *proxy*. Destina-se a evitar a violação *Cross Site Request Forgery* [10], onde não há um controlo de sessões e se permite que um utilizador efetue alterações numa sessão de um outro utilizador. Para evitar isto, antes de cada operação verifica-se se o estado da sessão atual php é coincidente com o parâmetro *sessionId*.

4.4.2.2 - *logout*:

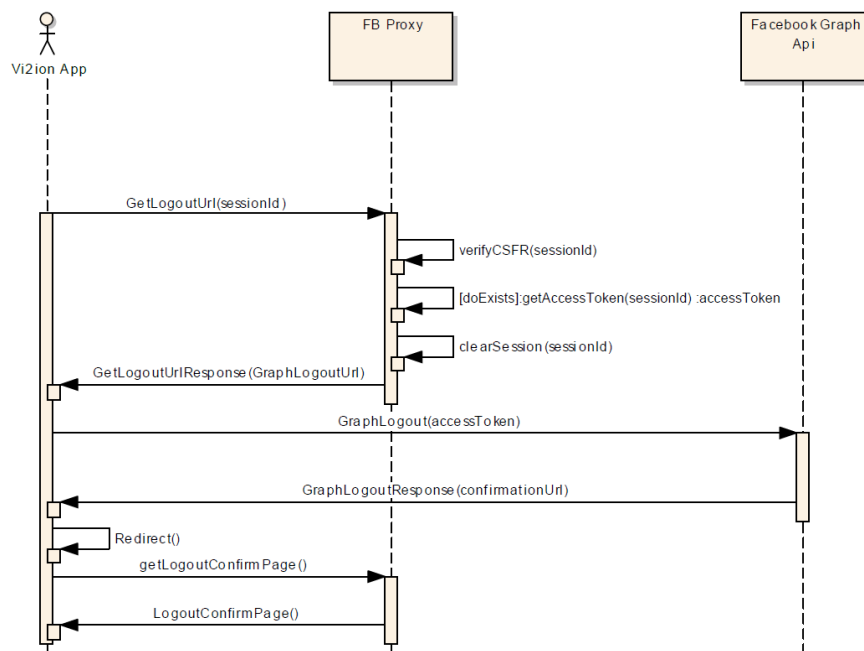


Figura 27: Sequência de Logout

A operação de *Logout* destina-se ao utilizador terminar a sessão. O processo inicial é eliminar a sessão da persistência, composta pelo *accessToken* e *sessionId*. De seguida é invocado o *logout* na Api do Facebook. Por fim, uma mensagem de confirmação é mostrada ao utilizador.

4.4.2.3 - *share, me, (Facebook Graph Api Call)*:

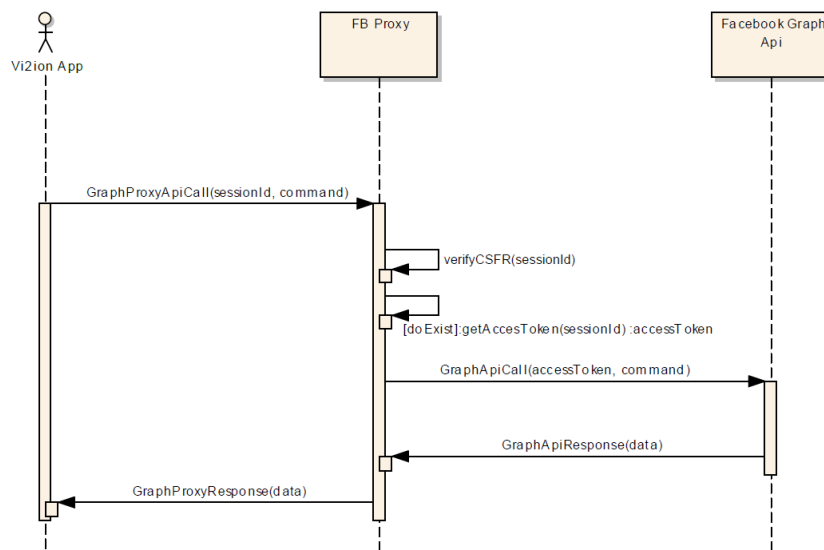


Figura 28: Sequência de uma chamada à *Graph Api*

A partir do processo de login apenas o *sessionId* é trocado entre o utilizador e a *proxy* para as chamadas à Graph Api, como ilustrado no diagrama. A Graph Api contém o conjunto de operações

que se podem fazer na rede social. A sua documentação é extensa [9], para este projeto apenas foi utilizada a função de share (*me/feed*) e consulta de detalhes básicos da conta pessoal (*me*). A operação a executar, e os seus argumentos foram simplificados no diagrama pelo parâmetro *command*.

Exemplo de uma visualização de um ecrã de *share*:

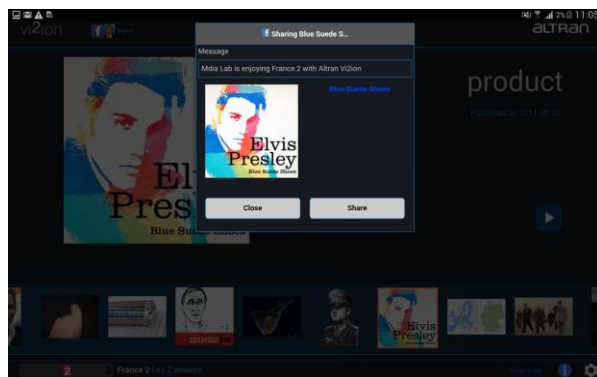


Figura 29: Ecrã de Share

4.5 - Vídeo OTT

4.5.1 – HTTP Adaptive Streaming

O vídeo *over-the-top* tem-se revelado uma tendência importante no sector do multimédia, e tem sido uma tecnologia que tem vindo a amadurecer ao longo do tempo. Fruto deste amadurecimento apareceu o conceito de *Adaptive Streaming Over Http*, corresponde ao ajuste dinâmico da *stream* de vídeo face às condições do meio, por exemplo: velocidade da rede, perda de pacotes, estado do *buffer*, estado local do *cpu* e memória. Esta adaptação consiste em ter o vídeo particionado em diversos períodos *Period*, cada segmento possui diversos contextos de adaptação (*Adaptation Sets*) para as diversas regiões, por sua vez cada *Adaptation Set* possui representações (*Representations*) de diferentes *bitrates* para cada período. Ainda cada representação está particionada em segmentos regulares (*Segments*). Desta forma em cada instante é selecionado a representação com o *bitrate* mais adequado ao meio, esta transição ocorre de forma fluida. Para representar esta meta-informação é necessário um ficheiro de manifesto que traduza o mapeamento de cada vídeo (MPD), dos seus segmentos, e das suas representações e períodos. Este ficheiro pode ser generalizado pela seguinte figura [11]:

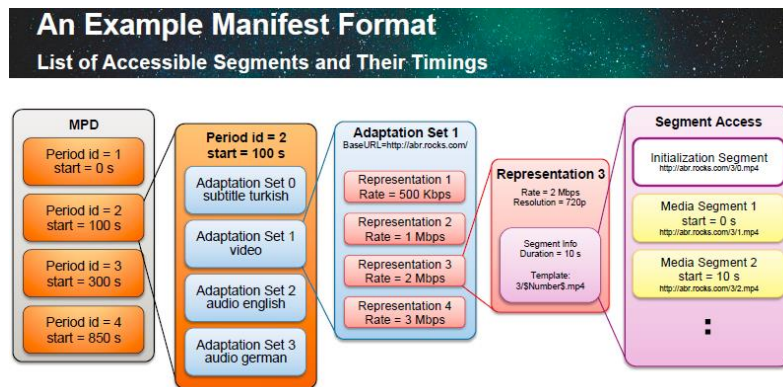


Figura 30: Esquema de um MPD (*Media Presentation Description*)

Diversas soluções proprietárias surgiram para endereçar o mercado, as maiores são: Microsoft Smooth Streaming [12], Apple HLS [13], Netflix [14], Adobe HDS [15], e Echostar [16], entre outras. Com esta proliferação as próprias marcas e a comunidade Mpeg decidiram avançar no sentido de criar um *standard* definitivo para o *Adaptive Streaming* por *Http*.

Neste contexto surge a norma Mpeg-Dash [17], uma solução de transporte *standard* para uma *stream* adaptativa sobre o protocolo *Http*. O scope deste mecanismo é o transporte de média (figura [11]), como tal é agnóstico a *codec*s, ao tipo de média, ao algoritmo de DRM etc. Como assenta no *Http* beneficia da infraestrutura já existente como Web cache, Cdn's, Cloud etc.

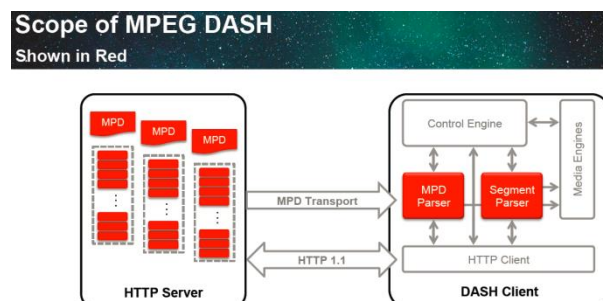


Figura 31: Scope do Mpeg Dash

4.5.2 - Mpeg-Dash-AVC/264 e player

O Mpeg-Dash-AVC/264 surge como uma extensão à especificação Mpeg-Dash como uma solução completa de vídeo HTML. A especificação propõem o *codec* de vídeo H.264/AVC e o *codec* de áudio HE-AAC.

Os Mpeg Dash *player* comerciais existentes no mercado são: Aricent, bitdash, BuyDRM, castlabs, Digital Primates, InterDigital, VisualOn, Qualcomm e RealNetworks. [40]

Dadas as restrições tecnológicas do projeto, nomeadamente tecnologia Javascript, *open-source* e sem custos mas ao mesmo contendo as funcionalidades mínimas necessárias para a aplicação, o *player* escolhido foi o *dash.js* do grupo DashIF [18]. Este cliente DASH é a base de todas as soluções comerciais enumeradas anteriormente, aproveita as extensões MSE (*Media Source Extensions* [19]) para a construção da *stream* de média e EME (*Encrypted Media Extensions* [20]) para DRM. Contudo

estas especificações ainda se encontram em fase experimental e não estão massificadas, só se encontram no Chrome versão 23, e no Internet Explorer versão 11.

As limitações deste *player* encontram-se detalhadas na secção 5.2.1.1

4.5.3 - Cliente DASH vi2ion

Do lado do cliente existe a implementação do *player* Dash.js, este poderá reproduzir média que venha associado a qualquer referência VOD. Cada *Presenter MainContentView* possui uma referência a um *sub-presenter OTTView*. A versão utilizada foi o Dash.js v1.1.2 [39].

OTTView:Backbone.View
<ul style="list-style-type: none">-tagName-template-dashPlayer-srcMPD-srcHQ-srcMQ-srcLQ-comm:CommunicationHandler
<ul style="list-style-type: none">+render+postRender+playAsset+unrender

Figura 32: Backbone Presenter OTTView

A fonte prioritária é o *srcMPD*, contudo pode dar-se o caso da falha da instanciação do *dashPlayer* se a *web-view* nativa não possuir as extensões MSE e EME. Nesse caso de exceção a *OTTView* vai analisar as condições da rede, e reproduzir a fonte adequada dentro das existentes (*srcHQ*, *srcMQ*, *srcLQ*) por via da *tag* de vídeo do Html5. Os formatos suportados então por este Presenter são então o mpd (condicionado), mp4, ogg e webm. Após a seleção da fonte adequada, o vídeo é reproduzido na sua *View*.

Exemplo da reprodução de um vídeo Mpeg-Dash em contexto VOD, associado à referência “*Top Gear*”:

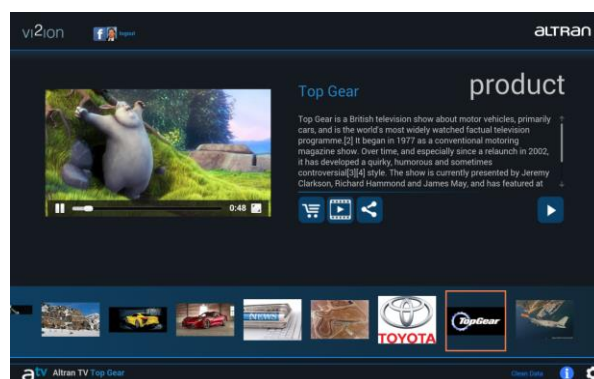


Figura 33: Reprodução de um vídeo Mpeg Dash

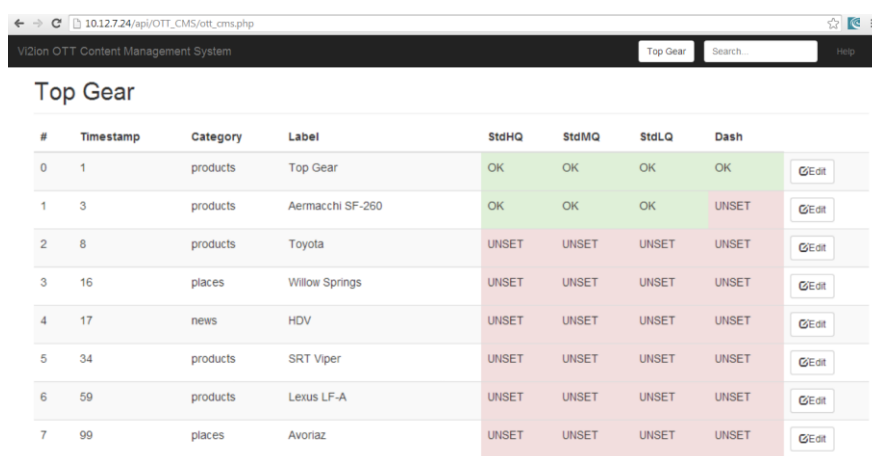
4.5.4 - Servidor DASH vi2ion

Do lado do servidor disponibiliza-se o ficheiro de manifesto MPD e o vídeo fragmentado pelas diversas representações. Para o *encoding* utilizou-se a ferramenta DASHEncoder [21] para fragmentar o vídeo e gerar o MPD. Para efeitos de demonstração e devido às limitações de tempo, o *encoding* foi feito apenas a um vídeo.

Do ponto de vista da aplicação os vídeos Mpeg-Dash estão presentes no modo VOD (Scope da MaqueteV2), e poderão surgir associados às referências existentes. Existe ainda a limitação do *web-view* nativa do dispositivo, não suportar as especificações MSE ou EME, nesse caso é necessário fazer *fallback* para a reprodução normal de um ficheiro **.mp4* com outra fonte. Estas extensões apesar de já constarem no w3c, ainda não estão massificadas, nos dispositivos móveis apenas estão presentes no android 4.4 Kitkat, com o Chrome superior à versão 35.0.0 e no Windows Phone 8.1.

Foi criado um *backoffice* para a gestão de quais as referencias que possuem vídeo em Mpeg-Dash, e caso seja necessário, quais as fontes para o *fallback* em **.mp4*. O *backoffice* permite associar a uma referência um ficheiro MPD, e três ficheiros **.mp4* para *fallback*, correspondentes a alta, média e baixa qualidade.

Segue-se um exemplo da visualização do Backoffice para a gestão de vídeos OTT, e a sua associação às referencias VOD do vídeo *Top Gear*:



#	Timestamp	Category	Label	StdHQ	StdMQ	StdLQ	Dash	
0	1	products	Top Gear	OK	OK	OK	OK	Edit
1	3	products	Aermacchi SF-260	OK	OK	OK	UNSET	Edit
2	8	products	Toyota	UNSET	UNSET	UNSET	UNSET	Edit
3	16	places	Willow Springs	UNSET	UNSET	UNSET	UNSET	Edit
4	17	news	HDV	UNSET	UNSET	UNSET	UNSET	Edit
5	34	products	SRT Viper	UNSET	UNSET	UNSET	UNSET	Edit
6	59	products	Lexus LF-A	UNSET	UNSET	UNSET	UNSET	Edit
7	99	places	Avoraz	UNSET	UNSET	UNSET	UNSET	Edit

Figura 34: Visualização do Backoffice da Gestão de Vídeos

4.6 - Conclusão

A Maquete V3 foi desenvolvida no seguimento da Maquete V2, de forma a dar continuidade ao conceito e reforça-lo com novas *features* e com suporte a canais *live*. As novas *features* foram escolhidas segundo os critérios do estudo da sociedade *Second Screen Society*, que identificou o conjunto de *features* que uma aplicação de *second screen* idealmente deverá oferecer. A aplicação foi redesenhada segundo a *framework* Javascript Backbone.Js por ser bastante flexível facilitando a integração dos módulos de controlo existentes na Maquete V2, por fornecer uma separação logica das responsabilidades MVP com mecanismos existentes para o efeito, e por incluir um motor de *templates* Html bastante flexível. O que facilitou o desenvolvimento ao separar totalmente as responsabilidades de *design* da lógica de UI. Outro aspeto importante é o baixo acoplamento entre

as diversas camadas MVP do Backbone, o que permite que cada componente seja testada facilmente nos testes unitários.

Foi adicionada uma componente social que possibilitou a integração com o Facebook, foi uma componente que trouxe um desafio interessante devido ao facto de não existirem API's oficiais para aplicações híbridas. O que obrigou procurar uma alternativa para suportar esta funcionalidade. A alternativa escolhida foi o desenvolvimento de uma *Proxy*.

Foi ainda adicionada uma componente de vídeo OTT. Para o efeito seleccionou-se a tecnologia de vídeo *Adaptive Streaming* por ser uma tendência atual e por melhorar bastante a qualidade de entrega de vídeo, oferecendo uma melhor experiencia final ao utilizador. Dadas as restrições do projeto escolheu-se a tecnologia Mpeg Dash com o respetivo *player open-source dash.js* do grupo DashIf.

Mais importante do que as *features* referidas, foi adicionado o suporte a canais *live* da televisão francesa, para este efeito usou-se o serviço ACE em tempo real do parceiro *Leankr*. Este serviço monitoriza em tempo real as fontes de *close caption's* dos canais de TV, retira as palavras relevantes, e fornece conteúdo semanticamente relacionado com estas.

O balanço da MaqueteV3 foi bastante positivo a nível pessoal e coletivo (Altran), atualmente a Maquete V3 faz parte das demonstrações oficiais do grupo e está difundida para mais de 20 países.

5 Testes

5.1 Introdução

Devido à natureza iterativa da metodologia Agile, é frequente existirem rondas de testes intermédias para assegurar que cada módulo está a cumprir com os seus requisitos. Contudo na reta final do ciclo de vida do processo do desenvolvimento é fundamental uma fase final de testes. Esta fase serve para assegurar a consistência do produto como o resultado da integração de diversos módulos heterogéneos, evidenciar as falhas existentes, e averiguar em que medida a lista de requisitos foi cumprida.

Este processo foi dividido em duas abordagens, *Black-Box* e *White-Box*.

5.2 – Testes *Black-Box*:

Testes *Black-Box* (igualmente conhecido como “testes funcionais”) trata o *software* como uma caixa preta, sem qualquer conhecimento da sua estrutura e funcionamento interno. Os *inputs* dos testes são introduzidos através da interface do software, e espera-se que o output deste corresponda ao esperado em conformidade com os requisitos definidos. As vantagens deste método são: Fornece uma perspetiva externa do *software*, existe uma separação entre a visão do desenvolvedor e do utilizador final; Eficiente para grandes volumes de código com uma lógica complexa; Não necessita de conhecimento interno do sistema, podendo ser realizado manualmente por pessoas sem conhecimentos de programação.

Estes testes podem ser conduzidos por um processo manual, ou por via de automação. Ambas as formas foram executadas.

5.2.1 - Execução Manual

Para a execução manual tendo como referencia a tabela de Requisitos, verificou-se se a aplicação deu resposta ao que inicialmente foi estabelecido:

Tabela 9:Testes de Aceitação Manuais aos Requisitos

Feature type	ShortName	Pass/Fail
Live Support	Suport live channels	Pass
OTT(VOD) Support	OTT Vídeo - Low/High resolution support	Pass
	Backoffice to Manage OTT Vídeo Assets	Pass
Gamification	Gamming - Quiz	N/A
Social Interaction (Facebook)	Login	Pass
	Share	Pass
	Show Profile	Pass
	Proxy on Server Side	Pass
User Interface	New UI/UX	Pass
	Responsive Design with Grids	Pass
	Small Screens Support	Pass

Feature type	ShortName	Pass/Fail
V2-Code Refactoring	MVC - Backbone.js	Pass
Security	HTTPS	N/A

Quase todos os requisitos foram respondidos com sucesso, à exceção do requisito de *Gamification* que não foi implementado, e do requisito de segurança HTTPS que foi implementado parcialmente, apenas na troca de mensagens com a API do Facebook.

Dentro ainda deste teste avaliou-se a qualidade do serviço de *ACE* fornecido pelo parceiro Leankr. As referências retornadas apresentaram-se sempre com bastante qualidade, dentro do contexto da emissão de TV. Contudo observou-se que a cadência de referências é dependente do tipo de programa que se encontra em emissão, por exemplo em programas de notícias, ou *talk shows* a frequência com que as referências chegam é bastante dinâmica, mas dão-se casos que em certos programas chegam muito pouco frequentemente, estando longos minutos sem chegar qualquer referência. O critério que influencia esta variação de frequência ainda não foi identificado, e é um ponto a melhorar juntamente com o parceiro.

5.2.1.1 – OTT- Mpeg-Dash DashIf Player

O grupo DashIf e a sua comunidade possuem um documento aberto *DASH-AVC-264-Test-Vectors* [32] que visa a definição de vários *test cases* assim como os seus critérios de aceitação para o seu *player* DashIf em conformidade com a norma Mpeg-Dash-AVC/264.

Estes testes destinam-se apenas a testar as capacidades de reprodução do *player* DashIf *open-source* escolhido para o projeto deixando fora de foco a geração do conteúdo fonte, referenciado pelo MPD.

Para cada *test-case* os critérios gerais de aceitação são:

- A1. A partir do momento em que o MPD é acedido, o vídeo começa a reprodução em um tempo máximo de 5 segundos.
- A2. O vídeo representado pelo MPD é reproduzido até ao fim
- A3. Áudio e Vídeo são reproduzidos sem que se note alguma falha de sincronização e.g: *lip sync*
- A4. Se o MPD inclui várias representações e/ou períodos, a sua comutação ao longo da reprodução terá de ser impercetível.

Na documentação vários *test-cases* são propostos de forma a abranger a especificação Mpeg-Dash-AVC/264. A cada *test-case* está associado um *test-vector* com o MPD adequado ao cenário, estes *test-vector's* estão disponíveis na página do grupo *DashIf* [33]. O teste irá ser feito com o *player* integrado na aplicação vi2ion, instalada num *tablet* Samsung Galaxy Tab 10.1, com o Android 4.4.2, em que a *web-view* nativa suporta as especificações MSE e EME. Para este teste apenas se vão efetuar os *test cases* que se consideraram essenciais à reprodução de conteúdo:

1. Reprodução de um MPD com um único período, diversas representações de uma única resolução e diversas *bitrates*.

Neste cenário o cliente Dash acede a conteúdo VOD. O vídeo é fornecido no formato DASH-264/AVC. Dois *Adaptation Sets* são fornecidos: Um para o vídeo, em que os segmentos existem em intervalos regulares, com a mesma resolução espacial, mas *bitrates*

diversificados. O outro *Adptation Set* representa o áudio, este possui uma única representação no formato HE-AACv2. Para representar esta fonte foi utilizado um MPD em conformidade com este cenário [34].

Face aos critérios de aceitação definidos, obteve-se a seguinte tabela:

Tabela 10:Mpeg Dash Test Case 1

Acceptance Criteria	Result
A1	Pass
A2	Pass
A3	Pass
A4	Pass

Todos os critérios de aceitação foram cumpridos, este *test-case* foi aceite.

2. Reprodução de um MPD com um único período, diversas representações com diversas resoluções e diversas *bitrates*.

Neste cenário o cliente Dash acede a conteúdo VOD. O vídeo é fornecido no formato DASH-264/AVC. O *encoding* é feito na resolução máxima, mas também em resoluções menores, para explorar as otimizações resultantes do aumento da compressão na transmissão. Dois *Adaptation Sets* são fornecidos: Um para o vídeo, em que os segmentos existem em intervalos regulares, com a diferentes resoluções e diferentes *bitrates*. O outro *Adptation Set* representa o áudio, este possui uma única representação no formato HE-AACv2. Para representar esta fonte foi utilizado um MPD em conformidade com este cenário[35].

Face aos critérios de aceitação definidos, obteve-se a seguinte tabela:

Tabela 11:Mpeg Dash Test Case 2

Acceptance Criteria	Result
A1	Pass
A2	Pass
A3	Pass
A4	Fail

Observou-se no decorrer do teste que o critério A4 não foi totalmente cumprido, nas transições entre representações com diferentes resoluções, por vezes notou-se um pequeno *soluço*, que embora pequeno é suficiente para quebrar a noção de fluidez.

3. Reprodução de um MPD com múltiplos períodos, diversas representações com diversas resoluções e diversas *bitrates*.

Este *test-case* pretende reproduzir o cenário do *test-case* anterior, mas a representação é feita com diversos períodos, à semelhança de um cenário real de VOD de média ou longa duração. Para representar esta fonte foi utilizado um MPD em conformidade com este cenário [36].

Face aos critérios de aceitação definidos, obteve-se a seguinte tabela:

Tabela 12:Mpeg Dash Test Case 3

Acceptance Criteria	Result
A1	Fail
A2	Fail
A3	Fail
A4	Fail

Observou-se que em representações com múltiplos períodos, a reprodução do vídeo não foi conseguida, apenas se ouvia o som. Este *test-case* foi dado como falhado.

4. Reprodução de um MPD sem representações de vídeo, com múltiplas representações de áudio.

Este cenário foca-se somente na reprodução de áudio, e nas capacidades da especificação Mpeg Dash em adaptar a *stream* com este conteúdo. São utilizados dois *Adaptation Set's*, representações em diversos *bitrates*, usando o *codec* HE-AACv2. Para representar esta fonte foi utilizado um MPD em conformidade com este cenário [37].

Face aos critérios de aceitação definidos, obteve-se a seguinte tabela:

Tabela 13:Mpeg Dash Test Case 4

Acceptance Criteria	Result
A1	Pass
A2	Pass
A3	N/A
A4	Pass

Todos os critérios de aceitação foram cumpridos, este *test-case* foi aceite.

Observa-se que o *player* em estudo possui ainda algumas limitações em ambiente Android, consequência de ser uma tecnologia recente, e também pelo facto das api's da extensão MSE não se encontrarem completamente implementadas na *web-view* nativa no Android 4.4.2. Neste estado o único cenário aceitável é ter um vídeo representado por um único período, em que as representações idealmente terão de ser com a mesma resolução espacial, dando apenas margem para aplicar a compressão no *bitrate* do vídeo e ao nível do áudio. Para o scope das demonstrações do vi2ion este cenário é suficiente, mas caso se evolua para uma solução comercial será necessário por optar por uma solução Mpeg Dash comercial completa.

5.2.1.2 - Network Failure Handling

O vi2ion é uma aplicação móvel profundamente dependente da ligação à rede, como tal espera-se que as situações de falha de rede estejam previstas e sejam tratadas convenientemente. Para tal elaborou-se um diagrama de estados que ilustrasse esta situação, onde estão representados os vários ecrãs, em que cada um deles ocorre uma falha de ligação. O objetivo é que ao ocorrer a falha, a aplicação abra a página das definições com o feedback do sucedido, identificando qual a interface de rede ativa em que ocorreu a falha.

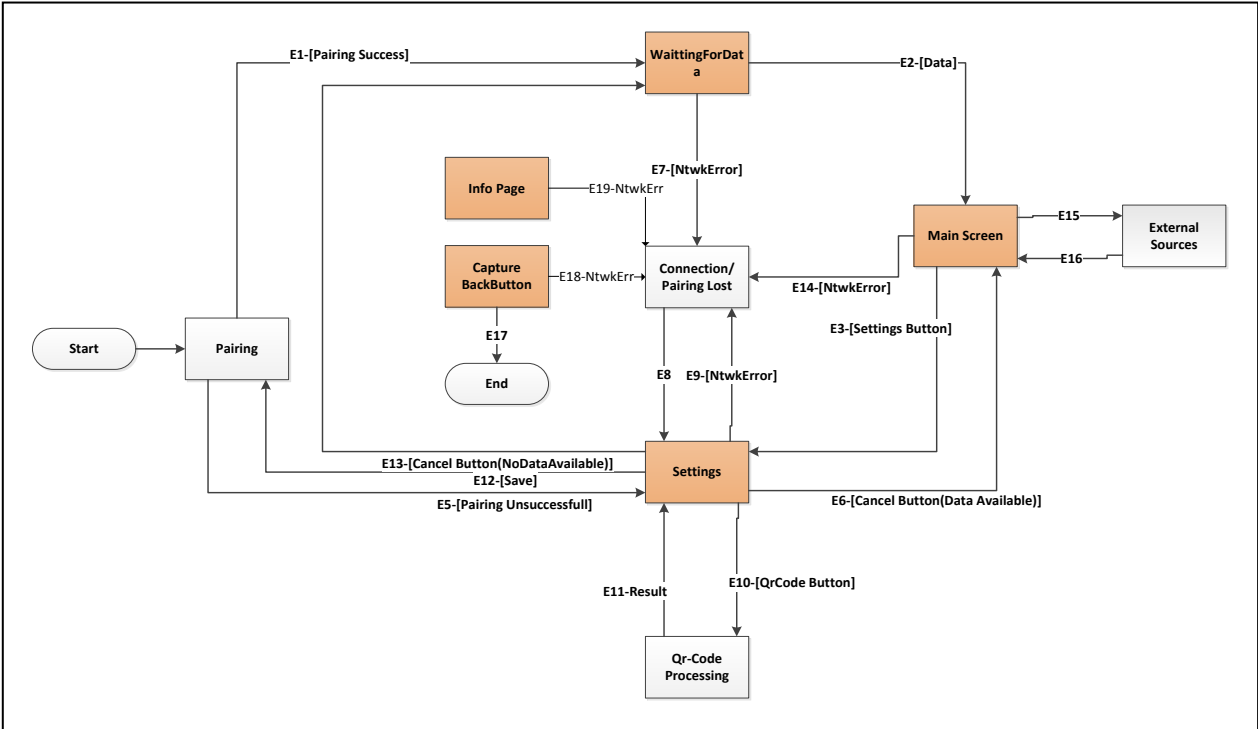


Figura 35: Diagrama de Transição de Ecrãs

As falhas de ligação são suscetíveis de serem detetadas nos ecrãs de “waiting for data”, “main screen”, “settings”, “capture back button” e na “Info”. As arestas correspondentes às falhas de ligação nestas janelas são respetivamente: E7, E14, E9, E18 e E19.

Transition	Open Configuration Menu
E7	Yes
E14	Yes
E9	Yes
E18	Yes
E19	Yes

Figura 36: Transições por Falha de Rede

5.2.2 - Execução Automatizada:

O dispositivo físico utilizado foi o Samsung Galaxy Note 8 com o Android 4.2, e a ferramenta para o método de testes *Black-Box* foi o *calabash* [22], uma ferramenta de automação de testes de aceitação para aplicações nativas Android e iOS. Utiliza a ferramenta de *Behaviour Driven Development* (BDD) Cucumber [23] para a descrição e definição dos testes. Esta ferramenta simula a interação humana através de *clicks* e verificações (*assertions*). Os testes e os seus passos são descritos em texto simples com uma linguagem natural, são agrupados por *Features* e por *Scenarios*:

Feature: ExternalSources
Scenario: ExternalSources
Given I Started Vision
Then I Start Hit Many References and check their external sources

Feature: SwitchObjectAndStreamMode
Scenario: SwitchObjectAndStreamMode
Given I Started Vision
Then I Start Hit Many References and on each i return to <i>stream</i> mode
Feature: GetInfo
Scenario: GetInfo
Given I Started Vision
When I am about to press Info button
Then I am presented with a page with project description, and i press OK to quit
Then I wait for 5 seconds
Feature: BackButton
Scenario: BackButton
Given I Started Vision
When I am about to press back button
Then I am presented with a page with exit confirmation, and i press OK to go back
Then I wait for 5 seconds
Feature: FacebookLoginAndShare
Scenario: FacebookLoginAndShare
Given I Started Vision
When I click Facebook Login icon
Then I see a Facebook Login Page
Then I place my credentials
When I submit my credentials
Then I see Successfull Facebook Response
Then I close the Facebook page
Then I see my profile photo on main screen
Then I share some reference on my Facebook wall

Figura 37: Descrição da Automação dos Testes

Nesta definição não foram cobertos todos os casos de uso por limitações de tempo, contudo efetuaram-se 5 *features* de utilização comuns:

ExternalSources: Representa a *feature* de aceder a uma fonte externa existente numa referência, o teste inicia a aplicação em modo *live* num canal aleatório, clica numa referência aleatória, abre a sua fonte externa, valida e fecha a fonte externa. Este passo é repetido um número pré definido de vezes.

SwitchObjectAndStreamMode: Representa a *feature* de comutar entre o modo Objeto e o modo *Stream*. O teste inicia a aplicação em modo *live* num canal aleatório, seleciona uma referencia aleatória, clica no botão para retomar o modo *Stream*, valida, e volta a clicar numa referência aleatória. Este passo é repetido um número pré definido de vezes.

GetInfo: Representa a ação de consultar os detalhes da aplicação. O teste inicia a aplicação em modo *live* num canal aleatório, clica no ícone de informações, valida o texto e sai.

BackButton: Representa a ação de carregar no *Backbuton* do Android. O teste inicia a aplicação em modo *live* num canal aleatório, clica no botão *Backbuton*, valida a mensagem de confirmação de que se deseja sair da aplicação, e fecha a notificação.

FacebookLoginAndShare: Representa a ação de efetuar login e depois fazer share do conteúdo de uma referencia aleatória. O teste inicia a aplicação em modo *live* num canal aleatório, clica no ícone de login do Facebook, valida o formulário de login, insere as credenciais válidas, submete as credenciais, valida a resposta de sucesso no *inAppbrowser*, fecha o *inAppbrowser*, seleciona uma referência aleatória, e faz share do seu conteúdo, por fim valida a notificação do share efetuado.

Cada step individual necessita de uma definição, esta é feita em Ruby e faz a ponte entre a descrição em texto simples, e a API em Rubi do Calabash para as chamadas nativas da ferramenta.

Ex: (test_steps.rb)

```
Given(/^I Started Vision$/) do
  @current_page = page(ViZionPage).await(timeout: 30)
  @current_page.goToMainScreen
end
```

Figura 38: Exemplo de Implementação de um Passo Automatizado -1

(viZionPage.rb)

```
class ViZionPage < Calabash::ABase
  def trait
    ".*"
  end
  def goToMainScreen
    wait_for_elements_exist(["CordovaWebView css:'#doSubmit'"], :timeout => Integer(15))
    if(query("CordovaWebView css:'#doSubmit'") == [])
      changeChnToRandomChannel()
    else
      touch(query("CordovaWebView css:'#doSubmit'"))
    end
    sleep(2)
  end
end
...
```

Figura 39: Exemplo de Implementação de um Passo Automatizado -2

Output:

```
Feature: ExternalSources

  Scenario: ExternalSources                                # features\ExternalReferences.feature:3
    5967 KB/s (543920 bytes in 0.089s)
    4477 KB/s (2961851 bytes in 0.646s)
    Given I Started Vision                                # features/step_definitions/login_steps.rb:1
    Then I Start Hit Many References and check their external sources # features/step_definitions/login_steps.rb:22

Feature: GetInfo

  Scenario: GetInfo                                        # features\GetInfo.feature:3
    Given I Started Vision                                # features/step_definitions/login_steps.rb:1
    When I am about to press Info button                   # features/step_definitions/login_steps.rb:6
    Then I am presented with a page with project description, and i press OK to quit # features/step_definitions/login_steps.rb:10
    Then I wait for 5 seconds                               # calabash-android-0.5.0.pre2/lib/calabash-android/steps/progress_steps.rb:10

Feature: BackButton

  Scenario: BackButton                                    # features\HitBackBtn.feature:3
    Given I Started Vision                                # features/step_definitions/login_steps.rb:1
    When I am about to press back button                   # features/step_definitions/login_steps.rb:14
    Then I am presented with a page with exit confirmation, and i press OK to go back # features/step_definitions/login_steps.rb:18
    Then I wait for 5 seconds                               # calabash-android-0.5.0.pre2/lib/calabash-android/steps/progress_steps.rb:10

Feature: SwitchObjectAndStreamMode

  Scenario: SwitchObjectAndStreamMode                    # features\SwitchObjectAndStream.feature:3
    Given I Started Vision                                # features/step_definitions/login_steps.rb:1
    Then I Start Hit Many References and on each i return to stream mode # features/step_definitions/login_steps.rb:62

Feature: FacebookLogin

  Scenario: FacebookLogin                                # features\FacebookLogin.feature:3
    Given I Started Vision                                # features/step_definitions/login_steps.rb:1
```

```

When I click Facebook Login icon      # features/step_definitions/login_steps.rb:26
Then I see a Facebook Login Page      # features/step_definitions/login_steps.rb:30
Then I place my credentials           # features/step_definitions/login_steps.rb:34
When I submit my credentials          # features/step_definitions/login_steps.rb:38
Then I see Successfull Facebook Response # features/step_definitions/login_steps.rb:42
Then I close the Facebook page        # features/step_definitions/login_steps.rb:46
Then I see my profile photo on main screen # features/step_definitions/login_steps.rb:54
Then I share some reference on my Facebook wall # features/step_definitions/login_steps.rb:58

5 scenarios (5 passed)
21 steps (21 passed)

```

Figura 40: Output dos Testes Automatizados

5.3 – Testes *White-Box*:

Testes *White-Box* (também conhecidos por “testes estruturais”) são testes que tiram partido do conhecimento interno da aplicação para condução do processo de testes. Esta metodologia de testes traz vantagens sobre a metodologia *Black-Box*, uma vez que possui uma maior profundidade, permite evidenciar anomalias que podem não ser diretamente perceptíveis ao utilizador através da UI. Existem dois níveis de testes *White-Box* [23], o primeiro são os testes unitários em que se testa cada componente isoladamente, o segundo nível são os testes de integração em que se testa a conjunção das diversas componentes. Embora os testes de integração sejam fundamentais, nesta fase por limitações de tempo, apenas se abordaram os testes unitários.

5.3.1 Testes Unitários:

Os testes unitários foram divididos em 3 grupos: Testes de chamada às API’s do servidor pela aplicação vi2ion, testes de geração de Modelos (Backbone.Models) e testes de geração de elementos DOM pelos *Presenters* (Backbone.View). Para a execução dos testes unitários utilizou-se a biblioteca Qunit [24].

Segue-se o resumo dos testes unitários:

Tabela 14: Execução dos Testes Unitários

Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.125 Safari/537.36	
Tests completed in 7459 milliseconds.	
109 assertions of 109 passed, 0 failed	
1. VOD Pairing: Correct Settings (0, 4, 4)	Rerun392 ms
2. VOD Pairing: Wrong Settings(Feedback Given) (0, 1, 1)	Rerun67 ms
3. Live Pairing: Correct Settings (0, 1, 1)	Rerun56 ms
4. Live Pairing: Wrong Settings (With Given Feedback) (0, 1, 1)	Rerun78 ms
5. Content Retrieving: VOD: GetData (0, 5, 5)	Rerun514 ms
6. Content Retrieving: Live: GetData (0, 3, 3)	Rerun267 ms
7. EPG Label: Live: Get Vídeo Info (0, 2, 2)	Rerun149 ms
8. EPG Label: VOD: Get Vídeo Info (0, 4, 4)	Rerun83 ms
9. Facebook: Get OAuth Login Url (0, 3, 3)	Rerun97 ms

10. Facebook: Share Content (0, 2, 2) Rerun3885 ms
11. Facebook: Get Profile Info (0, 7, 7) Rerun955 ms
12. Facebook: Get Profile Picture (0, 1, 1) Rerun72 ms
13. Facebook: Get OAuth Logout *Url* (0, 1, 1) Rerun168 ms
14. Backbone Model Layer (Model Factory): People Model (0, 12, 12) Rerun17 ms
15. VOD Pairing: Correct Settings (0, 4, 4) Rerun392 ms
16. VOD Pairing: Wrong Settings(Feedback Given) (0, 1, 1) Rerun67 ms
17. Live Pairing: Correct Settings (0, 1, 1) Rerun56 ms
18. Live Pairing: Wrong Settings (With Given Feedback) (0, 1, 1) Rerun78 ms
19. Content Retrieving: VOD: GetData (0, 5, 5) Rerun514 ms
20. Content Retrieving: Live: GetData (0, 3, 3) Rerun267 ms
21. EPG Label: Live: Get Vídeo Info (0, 2, 2) Rerun149 ms
22. EPG Label: VOD: Get Vídeo Info (0, 4, 4) Rerun83 ms
23. Facebook: Get OAuth Login *Url* (0, 3, 3) Rerun97 ms
24. Facebook: Share Content (0, 2, 2) Rerun3885 ms
25. Facebook: Get Profile Info (0, 7, 7) Rerun955 ms
26. Facebook: Get Profile Picture (0, 1, 1) Rerun72 ms
27. Facebook: Get OAuth Logout *Url* (0, 1, 1) Rerun168 ms
28. Backbone Model Layer (Model Factory): People Model (0, 12, 12) Rerun17 ms
29. Backbone Model Layer (Model Factory): Places Model (0, 9, 9) Rerun19 ms
30. Backbone Model Layer (Model Factory): News Model (0, 5, 5) Rerun15 ms
31. Backbone Model Layer (Model Factory): Products Model (0, 10, 10) Rerun16 ms
32. Backbone Model Layer (Model Factory): Definitions Model (0, 8, 8) Rerun16 ms
33. Backbone View Layer (HTML Generation): Person View (0, 6, 6) Rerun54 ms
34. Backbone View Layer (HTML Generation): Places View (0, 6, 6) Rerun50 ms
35. Backbone View Layer (HTML Generation): News View (0, 6, 6) Rerun91 ms
36. Backbone View Layer (HTML Generation): Products View (0, 6, 6) Rerun67 ms
37. Backbone View Layer (HTML Generation): Definitions View (0, 6, 6) Rerun115 ms

Os testes no intervalo [1;13] correspondem às chamadas à API do servidor do vi2ion por parte da aplicação, os testes do intervalo [14;18] correspondem à geração de modelos de cada tipo pelos dados que chegam do servidor. Por fim os testes do intervalo [19;23] correspondem às camadas *Presenter/View* e asseguram que foi gerado um elemento DOM corretamente a partir de um modelo *mock* respetivamente para cada tipo. Cada teste vem acompanhado com 3 dígitos, o dígito mais à direita (preto) corresponde ao número de verificações que foram efetuadas nesse teste, o dígito do meio (verde) corresponde ao número de verificações que foram efetuadas com sucesso e foram validadas, o dígito à esquerda (vermelho) corresponde ao número de verificações que falharam o processo de validação. Este plano de testes com as verificações feitas em detalhe encontra-se em anexo.

5.3.2 - Memory Management

Qualquer aplicação móvel tipicamente tem de manifestar uma maior preocupação com os recursos, uma vez que estes são mais limitados, acresce-se ainda o facto de ser uma aplicação híbrida de página única. Este tipo de aplicações de página única têm de ter um extremo cuidado com os *memory leaks*. Numa página tradicional sem transições de conteúdo com Javascript, as vistas são comutadas a pedido ao servidor, o servidor responde com uma página nova, e a *heap* do *browser* é reiniciada, assim os *memory leaks* raramente são uma preocupação. Pelo facto das aplicações Javascript não fazerem *refresh* na comutação entre páginas, se existirem referências circulares entre o DOM e objetos Javascript, estes não têm maneira de serem coletados pelo *garbage collector* e ficam indefinidamente na *heap* a ocupar espaço.

Existem vários padrões que fazem incorrer em referências circulares, uns são criados explicitamente. Outros são mais difíceis de detetar como referências circulares criadas indiretamente por *closures* [25,26]. É da responsabilidade do programador efetuar boas práticas e evitar *memory leaks*.

Numa primeira análise efetuou-se um *profiling* com as ferramentas de *developer* do Google Chrome. Deixou-se a aplicação correr durante um período de 5 horas, e observou-se o seguinte registo de alocação da memória *heap* e elementos do DOM.

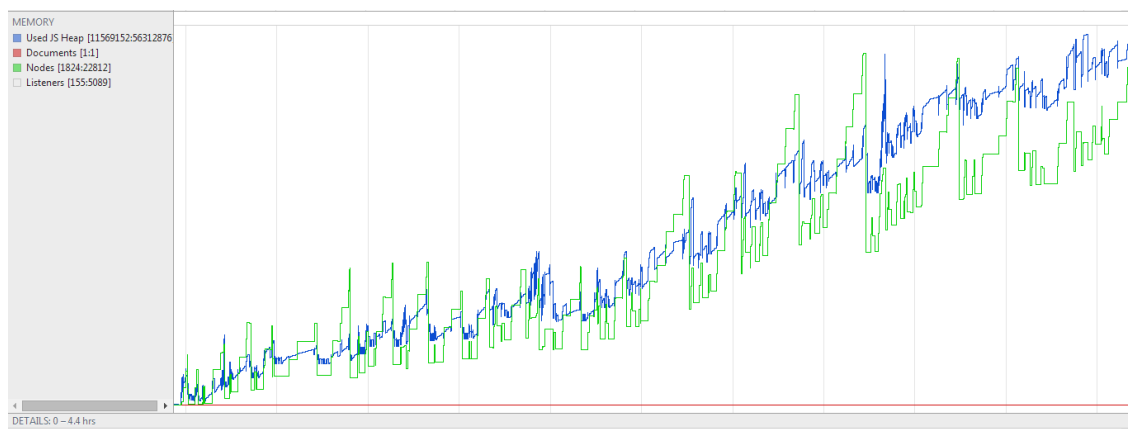


Figura 41: *Profiling* com Memory Leak

Observa-se uma clara tendência crescente, evidenciando a existência de *memory leaks*. Com as mesmas ferramentas é possível observar quais os elementos DOM *detached* (elementos que não estão presentes no DOM do documento, mas que o *garbage collector* não conseguiu eliminar da *heap*) obtendo assim uma pista para os *memory leaks* existentes. Para esta observação tiraram-se duas *heap snapshot's* (representação das entidades existentes na *heap*) no início da medição e no fim. E compararam-se as diferenças de elementos *detached* entre as duas *heap snapshot's*. Chegou-se ao seguinte resultado:

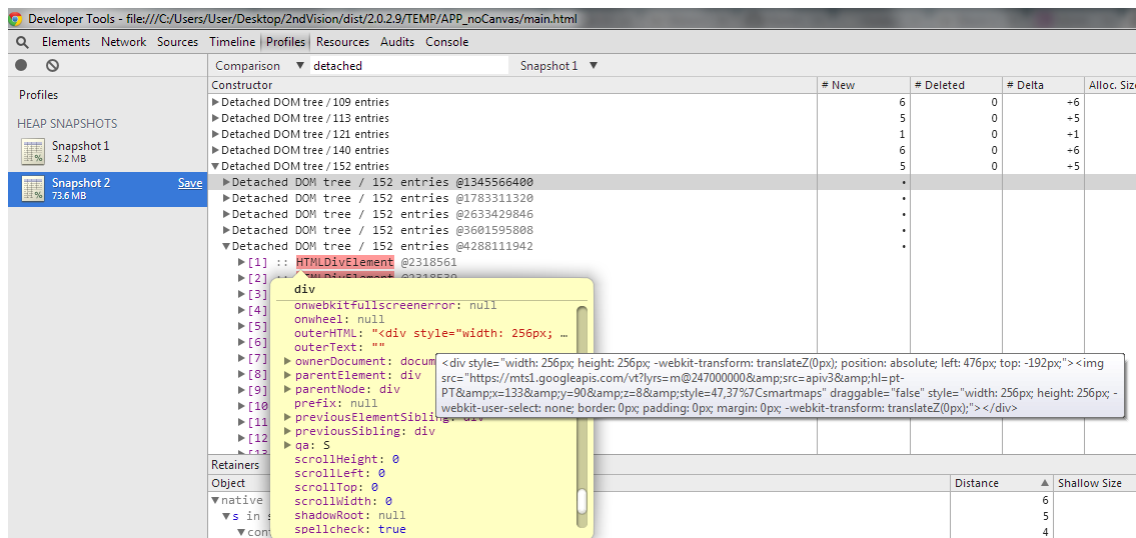


Figura 42: Detached Nodes no profiling com Memory Leak

Como previsto diversas entradas surgiram, no total excediam 1800 elementos. Uma observação detalhada demonstrou que a grande maioria continha no seu HTML referencias a um domínio relacionado com as API's do Goggle maps, dando a entender que se relaciona com as referências *Places* do vi2ion que utiliza a *widget* do Google Maps API, instanciada pelo *Presenter Backbone PlaceView.js*. Para confirmar esta teoria, desabilitou-se as referências relativas a *Places*, correu-se de novo a aplicação e efetuaram-se as mesmas medições:

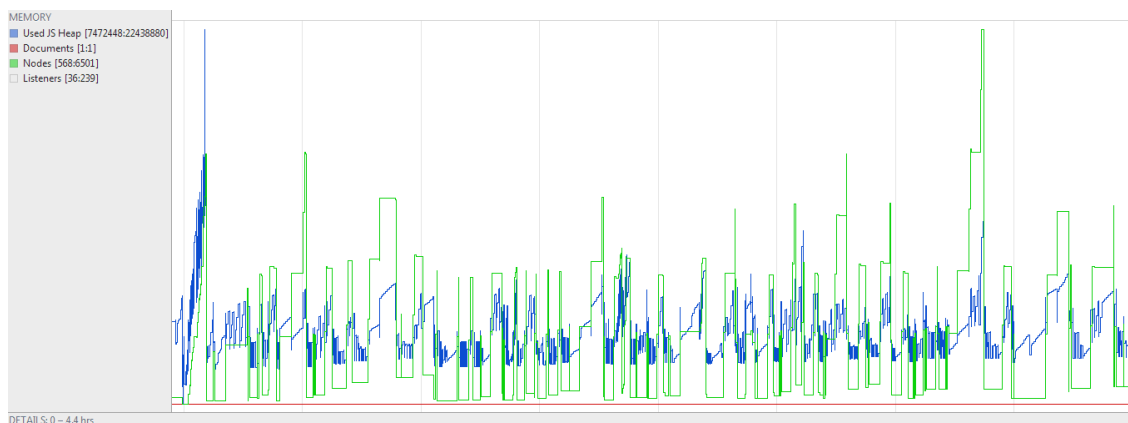


Figura 43: Profiling sem Memory Leak

	# New	# Deleted	# Delta	Alloc. Size	Freed Size	Size Delta
Constructor	7	11	-4	0	0	
▶ Detached DOM tree / 2 entries	2	3	-1	0	0	
▶ Detached DOM tree / 3 entries	2	1	+1	0	0	
▶ Detached DOM tree / 45 entries	1	1	0	0	0	
▶ Detached DOM tree / 5 entries						

Figura 44: Detached Nodes no profiling sem Memory Leak

Com estes resultados observa-se uma maior estabilidade na alocação da memória, e no número de elementos DOM existentes. O que se reflete no número de elementos DOM *detached*, que no fim da medição o número ficou pelos 50 elementos, ao invés dos 1800 registados no cenário anterior. Estas medições confirmam que a *widget* do Google Maps contém um *memory leak* internamente. Uma pesquisa na página do projeto revelou que este problema já havia sido reportado e foi reconhecido [27]. Contudo segundo o mesmo portal de *reporting*, até à data continua por resolver.

5.4 Conclusão

Para assegurar a qualidade da aplicação, e averiguar em que medida os requisitos foram cumpridos, efetuaram diversos testes. Os testes foram divididos em duas categorias: testes *White-Box* e testes *Black-Box*.

Para os testes *Black-Box* inicialmente fizeram-se uma série de testes funcionais que cobriram todos os requisitos definidos, duas abordagens foram tomadas: uma abordagem manual e uma automatizada. Na primeira abordagem manual testaram-se as funcionalidades implementadas foram todas passadas com sucesso. Contudo os requisitos de *Gamification* em que se pretendia implementar uma funcionalidade de quizzes, e o requisito não funcional de segurança HTTPS não foram cumpridos devido às limitações de tempo, sendo que o ultimo foi só aplicado parcialmente na comunicação com a entidade *Facebook Proxy*. Como estes requisitos haviam sido definidos como não sendo de prioridade crítica, o seu incumprimento não afetou o sucesso do projeto.

Dentro da abordagem manual foram ainda testados os cenários de perda de rede, e as capacidades de reprodução do *player* Mpeg-Dash. Este revelou algumas limitações em condições específicas de o MPD conter mais do que um período e mais do que uma resolução. Face às limitações do projeto, optou-se por um *player open-source* gratuito, que naturalmente mostra algumas limitações, contudo é possível demonstrar o conceito fazendo apenas variar as representações na *bitrate*. Para efeitos de demonstração cumpre com o requisito, mas caso se avance para uma solução comercial, será forçosamente necessário optar por uma solução comercial de um *player* MpegDash.

Os testes funcionais manuais não são indicados para o teste exaustivo de uma aplicação, por este motivo uma nova abordagem foi necessária. Automatizaram-se uma serie de testes a que se fizeram corresponder os cenários de utilização mais comuns. Para a automação dos testes utilizou-se a

ferramenta *calabash* de automação de aplicações android e iOS. Os testes automatizados não revelaram falhas.

Os testes *White-Box* pretenderam testar as componentes internas da aplicação. Testaram-se as diversas componentes de cada camada da implementação do modelo MVP. Os testes unitários não revelaram qualquer falha.

Dentro desta categoria foi ainda avaliada a gestão da memória. Foi detetado um *memory leak* na componente de UI da Api do Google Maps, concluiu-se que este foi um problema que já havia sido reportado, com um *ticket* aberto, mas que ainda não foi resolvido. Este comportamento não é crítico para utilizações curtas (<30 mins), não compromete uma demonstração da aplicação a um cliente, pelo que a componente em causa continua a ser utilizada.

Não sendo esta uma aplicação comercial, sendo uma aplicação de vendas, o balanço final dos testes foi positivo e a aplicação cumpriu com os requisitos.

6 Planeamento, Planeado vs. Atual

A Figura 45 representa o percurso planeado aquando da defesa intermédia. No decorrer do planeamento surgiram um conjunto de tarefas relacionadas com a *Mobile World Conference(MWC)* de 2014 em Barcelona. Os objetivos da Altran Portugal nesta exposição foram alcançados, e neste seguimento houve interesse do grupo Altran em divulgar internamente a plataforma vi2ion no seu estado atual. Para esta divulgação foi utilizado o *refactoring* da Maquete v2. O pacote Aplicação/Servidor foi divulgado, com a possibilidade de se utilizar um servidor público para o efeito, com controlo de acessos. Ainda tendo como referencia o planeamento inicial, o requisito “Modulo de Recomendações” não foi cumprido, pois não foi conseguido um acordo com o fornecedor pretendido pela Altran. Sem efeito este foi excluído do plano atual.

Os subsequentes requisitos do planeamento inicial em conjunto com os novos requisitos provenientes do contexto da exposição MWC formaram um novo planeamento, correspondente à Figura 46. As novas tarefas foram assinaladas a laranja.

A ordem de precedência das tarefas corresponde à prioridade atribuída pelo *Practice Manager*. Tendo em conta a tabela de requisitos, houve requisitos que não foram alcançados, nomeadamente a componente de *Gamification*, e a utilização do protocolo HTTPS na totalidade das trocas de mensagens entre a aplicação e servidor vi2ion. A componente de *Gamification* seria fornecida pelo atual parceiro Leankr como parte do serviço de ACE na forma de Quizes, contudo esta componente encontra-se temporariamente desabilitada pelo serviço do parceiro. O HTTPS por limitações de tempo apenas foi utilizado na troca de mensagens entre a Proxy Facebook e a Graph Api do Facebook. Da avaliação das prioridades feita com o *Practice Manager*, a estes requisitos foram-lhes atribuídas a prioridade Medium, como não sendo críticos para o sucesso do projeto.

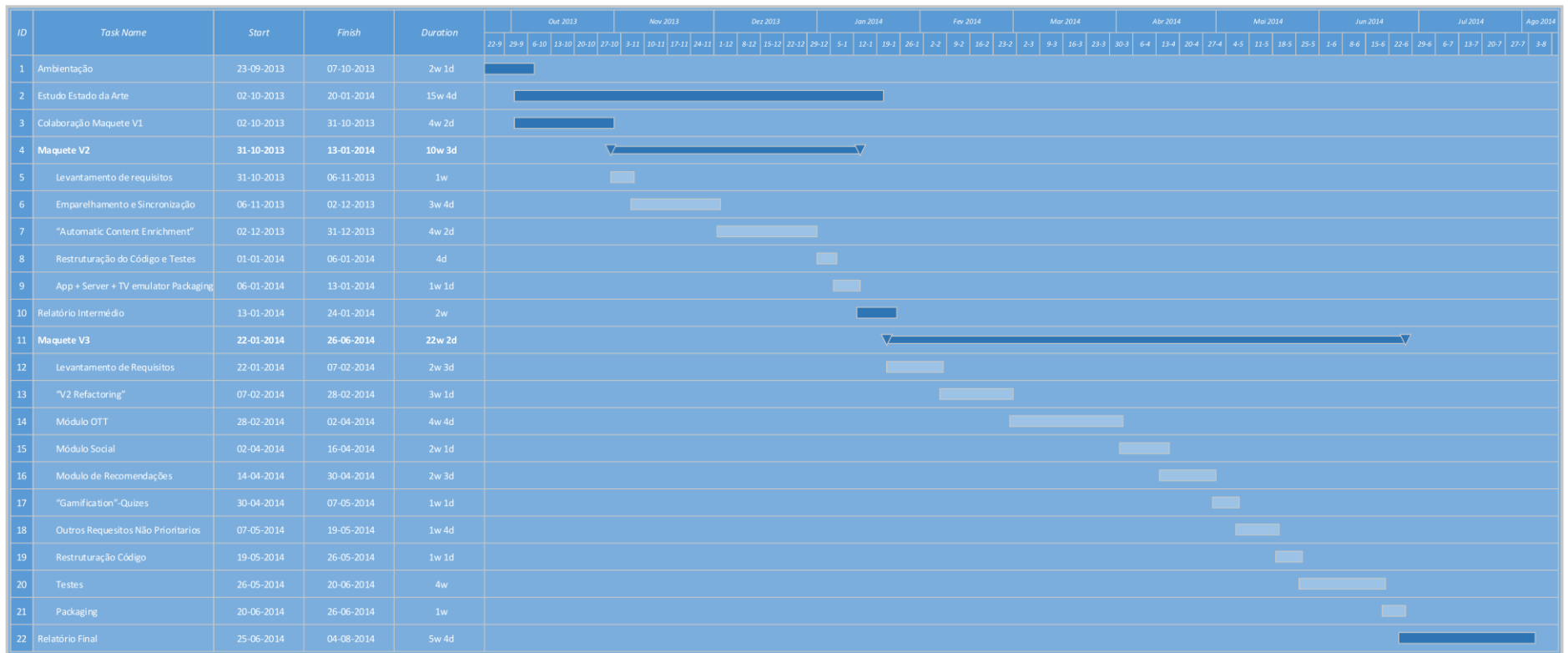


Figura 45: Planeamento da Defesa Intermédia 28/01/2014

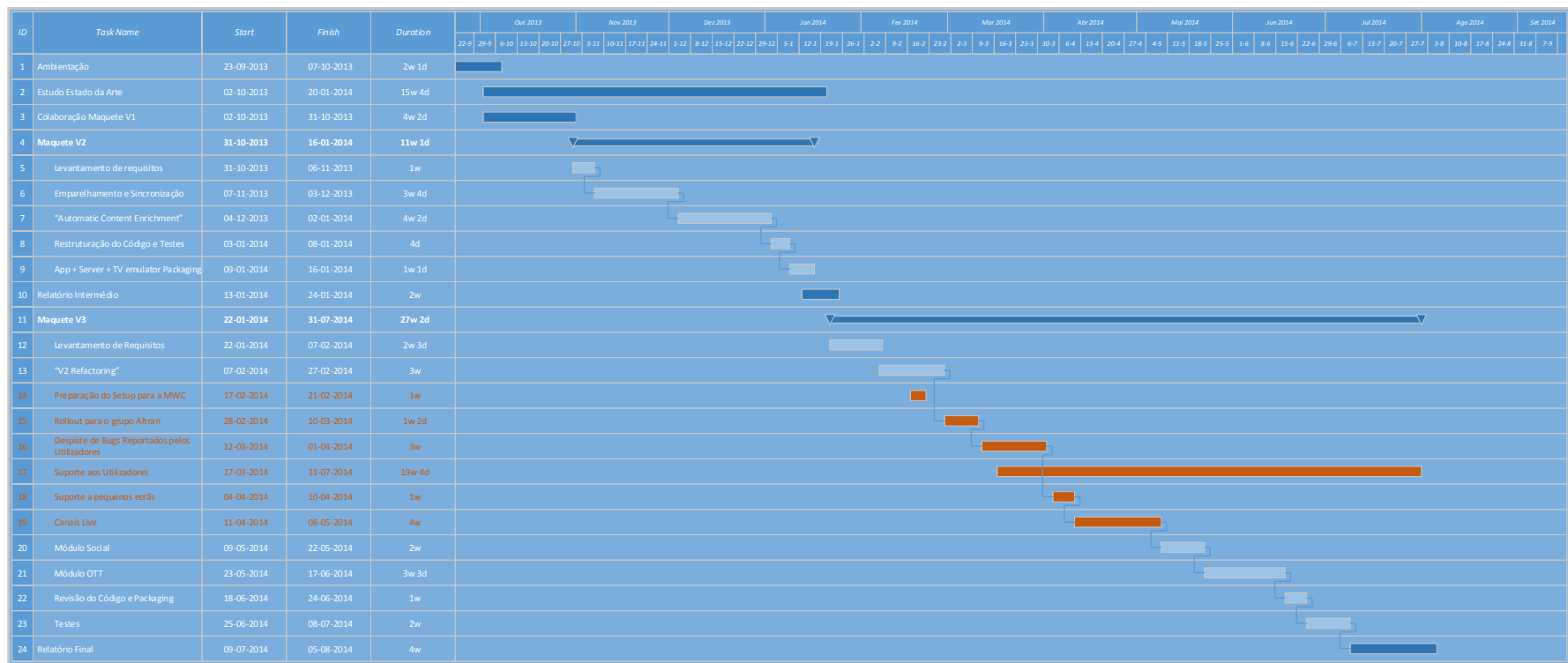


Figura 46: Planeamento Atual Gantt

6.1 Metodologia e Processos

A Altran Portugal tem vindo a adotar metodologias de desenvolvimento iterativas e incrementais inspiradas no modelo Agile. A adoção deste modelo tem em mente as frequentes alterações dos requisitos ao longo da fase de desenvolvimento, mantendo sempre um acompanhamento e uma avaliação constante da evolução global.

A metodologia utilizada caracteriza-se pelo seguinte diagrama:

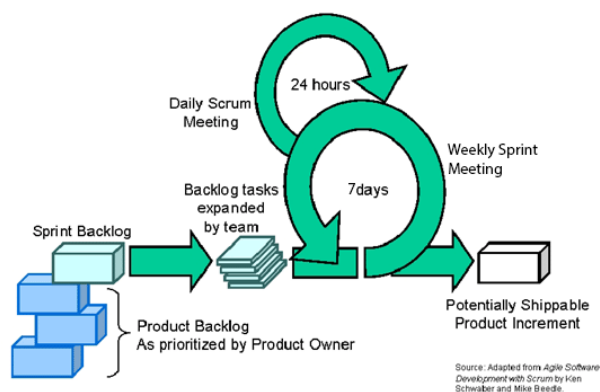


Figura 47: Diagrama Agile

Product Backlog:

Representa a lista de requisitos a atingir no âmbito do projeto. Os requisitos são enquadrados na metodologia e são ordenados pela prioridade atribuída pelo *Practice Manager*.

Sprint:

O Sprint consiste no período em que as equipas de desenvolvimento estão focadas na implementação de um conjunto de funcionalidades bem definido. No final do Sprint é esperado que esse conjunto de funcionalidades esteja alcançado, e que seja observado um incremento funcional no produto final.

Sprint Backlog:

Corresponde ao levantamento de funcionalidades existentes no *Product Backlog* para serem realizados no Sprint atual. A estimativa de esforço terá de estar em conformidade com a duração estipulada do Sprint.

Sprint Meeting:

Reunião com equipas de desenvolvimento envolvidas no Sprint. Nesta reunião participa o *Practice Manager*, o *Project Manager* da Altran Portugal com a sua equipa técnica, e o *Project Manager* da Altran SA com a sua equipa técnica. O objetivo da reunião é sincronizar o desenvolvimento das duas equipas, e planear os passos do Sprint atual.

Daily Meeting:

Reunião diária de curta duração com o Project Manager, em que se estabelecem as tarefas a realizar durante o dia, reportar obstáculos que tenham ocorrido no dia anterior para uma rápida resolução. Esta reunião pode ser feita por mail, caso o Project Manager não esteja disponível.

A nível de Processos, o desenvolvimento de cada Maquete é sustentado pelo sistema de controlo de versões Apache SVN, o cliente é ao critério do utilizador, neste caso foi usado o *turtle SVN*.

O desenvolvimento de cada Sprint é efetuado em *branch*, e após estar concluído com sucesso, é feito um *merge* do *branch* à linha principal de desenvolvimento *trunk*.

Na maquete v3 será usado um sistema de *bug-tracking* fornecido pela ferramenta *Jira*.

6.2 Riscos

Transversalmente à área, os riscos são uma realidade de qualquer projeto. É importante ter os riscos devidamente identificados e elaborar um conjunto de práticas que os minimizem. Particularmente na área de desenvolvimento de software, e no contexto deste projeto, os riscos são os seguintes:

Curva de Aprendizagem

Tendo em conta que o Estagiário não se encontrava familiarizado com as tecnologias necessárias ao desenvolvimento do projeto, a curva de aprendizagem é um risco real que pode em situação limite por em causa o sucesso de certas componentes. É da responsabilidade do Estagiário minimizar este risco, com o permanente estudo das tecnologias na literatura, e com a discussão de boas praticas com desenvolvedores experientes.

Divergências de Âmbito

A metodologia de desenvolvimento utilizada prevê mudanças moderadas de âmbito ao nível dos requisitos, contudo pode dar-se o caso de essa mudança ser tão profunda que coloca em causa a continuidade do projeto como este foi definido. Muitos fatores podem originar uma mudança drástica de rumo, desde decisões administrativas internas à empresa, exigências do cliente, até a fatores externos como o contexto socioeconómico.

Distribuição do Desenvolvimento de Módulos Intrinsecamente Dependentes

Neste caso concreto, na MaqueteV2 existem duas equipas de desenvolvimento que desenvolvem módulos distintos, Altran Portugal que desenvolve a aplicação móvel e a Altran SA que desenvolve o servidor. O sucesso global do projeto é pautado pelo sucesso das duas equipas, em que basta uma falhar, para todo o projeto falhar. O acompanhamento estreito definido na metodologia de desenvolvimento minimiza este risco, pondo as duas equipas em sincronismo permanente.

Limitações de Hardware

Sendo o público-alvo as utilizadoras de dispositivos móveis, as limitações de *hardware* são um risco que podem alterar a qualidade do produto final. É verdade que a tecnologia tem evoluído incrivelmente neste aspeto, mas continua a ser da responsabilidade de quem desenvolve que a aplicação consuma o mínimo de recursos para a função que desempenha.

7 Conclusão

O presente estágio surgiu com a motivação da Altran fornecer uma solução de *second screen* aos seus clientes. Este desafio foi dividido em diversas etapas, tendo começado pelo estudo do estado da arte.

Para endereçar o problema da concepção de uma aplicação de *second screen*, começou-se por categorizar o universo deste tipo de aplicações, para tal citou o estudo feito pela RedBee Media. Este estudo agrupou o universo das aplicações existentes em 3 grupos: *Dual Screening*, Atividade Síncrona, e *Companion Experience*. Cada grupo espelha a evolução da tecnologia, e vai sendo mais específico que o anterior, sendo que a tendência atual são as aplicações que se inserem no grupo de *Companion Experience*. Dada a relevância deste grupo, as suas aplicações também foram categorizadas em 4 quadrantes nos eixos “Onde” e “Como”: “Aplicações de Interação de Conteúdo”, “Aplicações de Controlo Remoto”, “Aplicações de Consumo de Conteúdo” e “Aplicações de Gestão de Conteúdo Remoto”. Face às tendências atuais o quadrante mais interessante é o “Aplicações de Interação de Conteúdo”, pelo que foi o quadrante escolhido para o vi2ion.

De seguida no estado de arte estudaram-se as componentes de uma aplicação do quadrante “Aplicações de Interação de Conteúdo”: A primeira etapa é a fase de sincronização com o sinal fonte (identificar e localizar temporalmente), e a segunda etapa é com os dados de sincronização obter dados de enriquecimento de conteúdo. Nas duas etapas foram estudados os respetivos estados da arte. Para a etapa de sincronização identificaram-se e compararam-se 3 métodos: *Watermarking*, *Fingerprinting* e *Timecoding*. Estudaram-se também as principais empresas/serviços existentes que baseiam as suas ofertas em cada um destes métodos. Uma vez que o alvo será implementar o serviço em *set-top-boxes* Hbbtv, a sincronização através de *Timecoding* é a indicada. O que não invalida que a requerimento de outros possíveis clientes se utilize um método de sincronização alternativo, dentro dos métodos estudados.

Para a fase da obtenção dos dados de enriquecimento estudaram-se 3 fornecedores, tendo-se escolhido a Leankr como fornecedor do serviço de ACE. O serviço da Leankr é endereçado ao mercado francês, pelo que é adequado ao primeiro cliente que motivou o início deste projeto.

Finalmente no estado da arte estudou-se o panorama atual das aplicações *second screen*. Aproveitando os critérios de seleção que a sociedade *second screen society* definiu como sendo *features* idealmente necessárias a uma aplicação *second screen*: “Controlo”, “Conteúdo Aumentado”, “Social”, “Discovery” e “Multi-Screen”, classificaram-se as aplicações estudadas. Observou-se que no panorama atual as *features* mais predominantes são as de “Discovery” e “Social”. A *feature* mais interativa “Conteúdo Aumentado” está pouco instalada e mostra margem para crescer. Pelo que o protótipo do vi2ion assumiu a orientação segundo as *features* “Conteúdo Aumentado” e “Social”.

Com as orientações do estudo do estado da arte em mente, prosseguiu-se à implementação do protótipo vi2ion. Este protótipo foi faseado e elaborado em duas maquetes incrementais, a MaqueteV2 e a MaqueteV3.

A MaqueteV2 foi o ponto de partida, foi desenvolvida como uma aplicação híbrida através do *middleware* IntelXdk, com tecnologias web Html5, Css3 e Javascript, com uma abordagem ao problema foi inspirada no *pattern* MVC. Esta maquete pretendeu demonstrar o conceito ACE do parceiro Leankr num contexto *offline* VOD, com 3 vídeos pré processados. Nesta maquete colaborou a equipa da Altran França na parte do servidor, e na simulação da *TV Player (set-top-box)*. Após a sua conclusão o resultado final foi exposto pela Altran na feira Mobile World Conference em Barcelona durante 4 dias.

A MaqueteV3 deu seguimento ao trabalho efetuado na MaqueteV2. A componente “Conteúdo Aumentado” foi reforçada com o suporte a canais *live* da televisão francesa, sendo ainda complementada com o suporte de vídeo OTT, através da tecnologia de Adaptive Streaming Mpeg Dash. A componente “Social” também foi dada como fundamental, pelo que também foi incluída nesta maquete. Dada a dimensão da aplicação justificou-se o uso de uma *framework* Javascript de modo a organizar o código, e aumentar a sua eficiência. A *framework* Backbone.js foi a escolhida para o efeito, e o *middleware* utilizado foi o Phonegap.

O desenvolvimento da MaqueteV3 concluiu-se com a etapa de testes de forma a validar o resultado final face aos requisitos. Os testes foram divididos em dois grupos: testes *Black-Box* e testes *White-Box*. Os testes *Black-Box* fizeram-se unicamente utilizando as interfaces de utilizador da aplicação, e foram feitos em duas abordagens: Numa abordagem manual onde se verificaram individualmente os requisitos, e se estes foram satisfeitos. Dentro desta abordagem tiveram especial foco os testes da aplicação em cenários de perda de rede, e também o comportamento do *player open-source* Mpeg-Dash. Verificou-se que o *player open-source* escolhido possui em situações específicas algumas limitações, e que caso a aplicação evolua para uma solução comercial, será necessário trocar o *player open-source* por um outro *player* comercial de Mpeg Dash. De modo a testar a aplicação de forma exaustiva, optou-se por uma abordagem automatizada, com recurso à ferramenta Calabash de automação de testes a Android e iOS. Os testes de aceitação automatizados não revelaram falhas.

Seguiram-se os testes *White-Box*, testes unitários onde se testaram individualmente os componentes de cada camada da aplicação, estes decorreram sem detetar nenhuma anomalia. Contudo foi detetado um *memory leak* nas *widgets* do Google Maps, este bug já havia sido reportado no portal de *reporting* de api's da Google, e atualmente continua por resolver.

As anomalias encontradas não foram consideradas críticas para o scope da aplicação. Como esta se destina a vendas e não a uso comercial, os requisitos foram considerados como satisfeitos.

O balanço global do projeto foi bastante positivo a nível pessoal e coletivo (Altran), atualmente a Maquete V3 faz parte das demonstrações oficiais do grupo e está difundida para mais de 20 países. A nível pessoal foi enriquecedor pois possibilitou o contacto com tecnologias web, em conjugação com tecnologias mobile, esta conjugação criou um conjunto de desafios e um paradigma totalmente novo à experiência pessoal.

Seguem-se algumas amostras da aplicação final, que ilustram as referencias retornadas pelo parceiro de ACE assim como as outras funcionalidades da aplicação.

Exemplo de uma referência *People* “Stevie Wonder” ocorrente no canal “France 2”, no programa “*Tout le monde veut prendre*”:

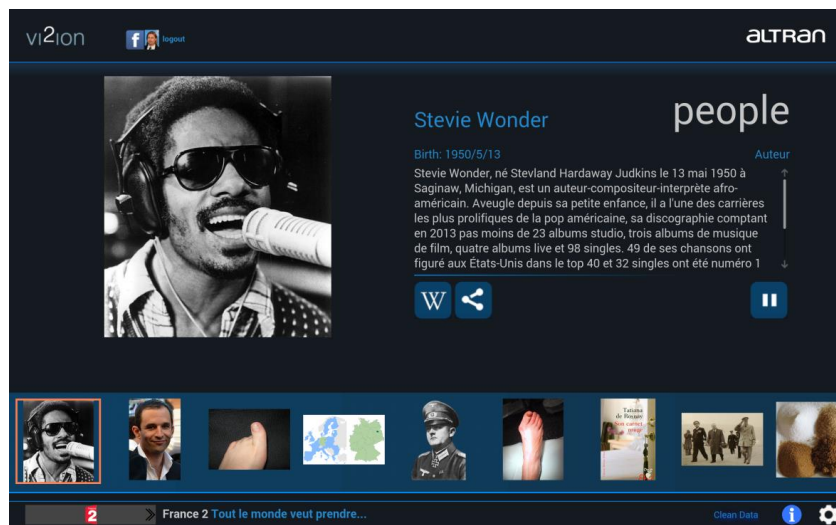


Figura 48: Exemplo de uma Referência *People*

Outros exemplos:

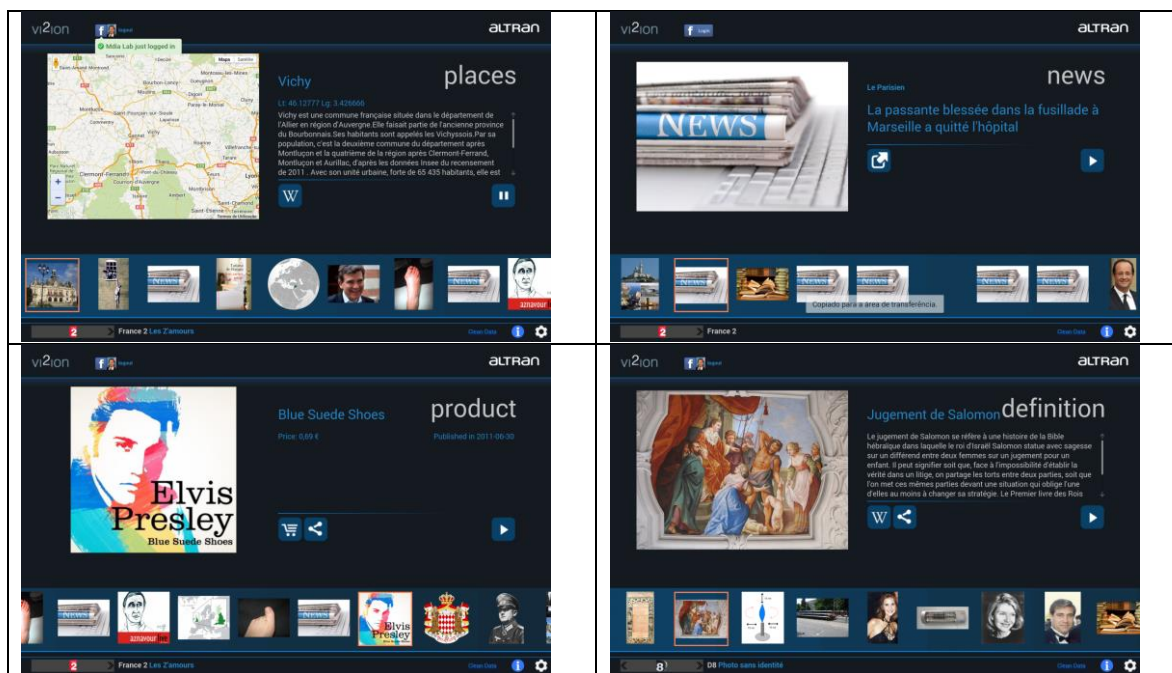


Figura 49: Exemplos de Referências

8 – Referências

Nota: Todas as referencias web foram verificadas à data de 16/08/2014.

8.1 - Capítulo 1:

- [1] - Telebriefing: Ericsson Mobility Report
- [2] - The Nielsen Company. (2012). State of the Media: The Cross-Platform Report
- [3] - Ericsson consumerlab: TV AND VÍDEO An analysis of evolving consumer habits
- [4] - <http://digitalvideospace.blogspot.pt/2014/01/monetizing-2nd-screen-business-models.html>
- [5] - <http://www.mediapost.com/publications/article/154542/#axzz2EaXrWdE>
- [6] - <http://www.mediaweek.co.uk/news/1110944/zeebox-strikes-major-investment-deal-Sky/>
- [7] - <http://money.cnn.com/2012/11/19/technology/innovation/getglue-viggle/index.html>

8.2 - Capítulo 2:

- [2] - <http://www.broadbandtvnews.com/2013/09/06/iptv-operators-gaining-market-share/>
- [3] - Special Eurobarometer 396 ; E-COMMUNICATIONS HOUSEHOLD SURVEY
- [4] - Audio Watermarking and Fingerprinting, for which Applications ? Leandro C.T., Pedro Cano, Eloi Batlle
- [5] - A Television Channel Real-Time Detector using Smartphones, Igor Bisio, Alessandro Delfino, Fabio Lavagetto
- [6] - ISO/IEC 13818-1:2007
- [7] - TV Everywhere Growth, Solutions, and Strategies North America (2nd Ed.) - Parks Associates - April 2012
- [8] - The 2nd Screen Society: Transforming vídeo consumption
- [9] - <http://www.mufin.com/products/audioid-live-channel-detection/>
- [10] - <http://www.egonocast.com/>
- [11] - <http://www.intonow.com/ci/soundprint>
- [12] - <http://www.civolution.com/applications/media-interaction/>
- [13] - <http://intrasonics.com/>
- [14] - www.leankr.com
- [15] - www.senseeive.com
- [16] - www.boxfish.com
- [17] - <http://get.viggle.com/>
- [18] - <http://getglue.com/>
- [19] - <http://gomiso.com/>
- [20] - <http://zeebox.com/>
- [21] - <http://shufflr.tv/>
- [22] - <http://www.intonow.com/ci>
- [23] - Google Play Id: com.sonypictures.hannibal
- [24] - <http://itunes.apple.com/ca/app/walking-dead-walkers-kill/id498529661>
- [25] - <http://meo.pt/conhecer/tv/meogo-fora-de-casa>
- [26] - Google Play Id: com.att.android.uverse
- [27] - <https://www.fan.tv/>

Outras referências que foram relevantes para o estudo do *Capítulo 2*, mas que não foram explicitamente citadas:

- Second Screen Series Paper 1 Whitepaper Red Bee Media
- Second Screen, SnellGroup
- AppscendSecondScreenWhitePaper
- JWT Intelligence-10 Ways Marketers Are Using the Second Screen
- Audio Watermarking and Fingerprinting, for which Applications ? Leandro C.T., Pedro Cano, Eloi Batlle

- A Television Channel Real-Time Detector using Smartphones, Igor Bisio, Alessandro Delfino, Fabio Lavagetto
- CONTENT FINGERPRINTING FROM AN INDUSTRY PERSPECTIVE, Craig Seidel

8.3 - Capitulo 3,4,5:

- [1] - Learning JavaScript Design Patterns, Addy Osman - O'Reilly
- [2] - <http://www.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle/>
- [3] - <http://www.funnyant.com/choosing-javascript-mvc-framework/>
- [4] - <https://www.todomvc.com>, <https://github.com/tastejs/todomvc>
- [5] - <https://developers.facebook.com/docs>
- [6] - The OAuth 2.0 Authorization Framework, <http://tools.ietf.org/html/draft-ietf-oauth-v2-31>
- [7] - <http://www.w3.org/TR/XMLHttpRequest/>
- [8] - <https://developers.facebook.com/docs/Facebook-login/manually-build-a-login-flow/v2.0>
- [9] - <https://developers.facebook.com/docs/graph-api/reference/v2.0>
- [10] - <http://shiflett.org/articles/cross-site-request-forgeries>
- [11] - HTTP Adaptive Streaming: Principles, Ongoing Research and Standards ICME 2013 – San Jose, CA
- [12] – <http://www.iis.net/expand/SmoothStreaming>
- [13] – <http://tools.ietf.org/html/draft-pantos-http-live-streaming>
- [14] – <http://www.netflix.com>
- [15] – <http://www.adobe.com/products/hds-dynamic-streaming.html>
- [16] – <http://www.echostar.com/>
- [17] – ISO/IEC 23009-1
- [18] – <http://dashif.org/>
- [19] – <https://dvcs.w3.org/hg/html-media/raw-file/tip/media-source/media-source.html>
- [20] – <http://www.w3.org/TR/encrypted-media/>
- [21] – <https://github.com/sleederer/DASHEncoder>
- [22] – <http://calaba.sh/>
- [23] – <http://www.chaudhary.org/White-Box.pdf>
- [24] – <http://qunitjs.com/>
- [25] - <http://www.ibm.com/developerworks/library/wa-memleak/>
- [26] - <http://www.javascriptkit.com/javatutors/closuresleak/index3.shtml>
- [27] - <https://code.google.com/p/gmaps-api-issues/issues/detail?id=3803>
- [28] - <http://www.nngroup.com/articles/ten-usability-heuristics/>
- [30] - <http://stackoverflow.com/questions/186244/what-does-it-mean-that-javascript-is-a-prototype-based-language>
- [32] - <http://dashif.org/w/2014/01/DASH-AVC-264-Test-Vectors-v09-CommunityReview.pdf>
- [33] - <http://dashif.org/testvectors/#SRMR>
- [34] - <http://dash.edgesuite.net/dash264/TestCases/1a/qualcomm/1/MultiRate.mpd>
- [35] - <http://dash.edgesuite.net/dash264/TestCases/2a/qualcomm/2/MultiRes.mpd>
- [36] - <http://dash.edgesuite.net/dash264/TestCases/5b/1/manifest.mpd>
- [37] - http://dash.edgesuite.net/dash264/TestCases/3a/fraunhofer/aac-lc_stereo_without_video/ElephantsDream/elephants_dream_audio_only_aac_stereo_sidx.mpd
- [38] - <http://tvnlive.dashdemo.edgesuite.net/live/manifest.mpd>
- [39] - <https://github.com/Dash-Industry-Forum/dash.js>
- [40] - <http://dashif.org/clients/>