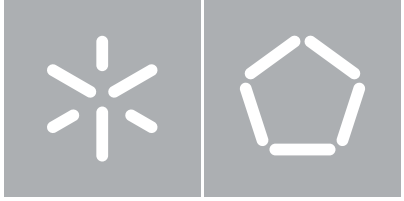
Universidade do Minho
Escola de Engenharia

Joaquim Alberto da Costa Anacleto

Desenvolvimento de uma aplicação web para dispositivos móveis - Monitorização e controlo de uma rede de digital signage

Junho de 2012



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Joaquim Alberto da Costa Anacleto

Desenvolvimento de uma aplicação web para dispositivos móveis - Monitorização e controlo de uma rede de digital signage

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de

Professor Doutor José Francisco Creissac Campos
Doutor Helder Pinto

Junho de 2012

This work was partly funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT -Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-015095.

Agradecimentos

Gostaria de agradecer ao meu orientador, Professor Doutor José Creissac Campos, pelo seu empenho, disponibilidade e conhecimento que foram fundamentais para a conclusão desta dissertação. Queria manifestar também o meu agradecimento à empresa Ubisign que me deu a oportunidade de realizar o projeto que tinha proposto. Em especial, fica o meu agradecimento ao meu orientador na empresa, Helder Pinto, pelo empenho e disponibilidade demonstrada, que foi fulcral para a conclusão com sucesso do projeto. Finalmente, agradeço aos meus colegas, familiares e amigos que me apoiaram e incentivaram durante a realização desta dissertação.

Abstract

The development of applications for smartphones is a relatively recent subject, yet it has been growing at a fast pace. It is an area that arises in a natural way because of the evolution of smartphones in terms of its characteristics, and also because of its growing popularity. The large communities behind the various platforms in the market (Android, iOS, etc) also help this area to be atractive and with plenty of success stories. Although initially the development of applications was done natively, there is another emerging trend that is developing smartphone applications using web technologies such as HTML5, CSS3 and JavaScript. This new trend is mainly driven by the implementation of HTML5 which has many new features, that allow greater integration of applications with devices.

This dissertation addresses the development of web applications for smartphones. The dissertation reviews the state of the art in web technologies and languages. Frameworks that speed up the development process will also be addressed. An approach to develop web application able to provide an interaction experience similar to that of native applications is proposed through an example. The project that was the basis for this dissertation was the development of a web application designed and adapted for smartphones, which will assist in monitoring and controlling a digital signage network.

Keywords: user interface, mobile applications, web development, design, usability.

Resumo

O desenvolvimento de aplicações para *smartphones* é uma área relativamente recente, mas que tem vindo a crescer a um ritmo bastante rápido. É uma área que surge de uma forma natural devido à grande evolução das características dos *smartphones* e também devido à sua crescente popularidade. As grandes comunidades por trás das várias plataformas do mercado (Android, iOS, etc) também ajudam a que esta seja uma área bastante apelativa e com bastantes casos de sucesso. Embora inicialmente só se desenvolvessem aplicações nativas, está a surgir outra vertente que é o desenvolvimento de aplicações para *smartphones* usando tecnologias web tais como o HTML5, JavaScript e CSS3. Esta nova vertente é impulsionada principalmente pela implementação do HTML5 que, com as suas muitas funcionalidades, permite uma maior integração das aplicações com os dispositivos.

Esta dissertação aborda o desenvolvimento de aplicações web para *smartphones*. É analisado o estado da arte das tecnologias e linguagens web. *Frameworks* que aceleram o processo de desenvolvimento também serão abordadas. Uma abordagem para desenvolver aplicações web capaz de fornecer uma experiência de interação semelhante à de aplicações nativas é proposto através de um exemplo. O projeto que serviu de base a esta dissertação foi o desenvolvimento de uma aplicação web, concebida e adaptada para ser utilizada em *smartphones*, que irá servir para ajudar na monitorização e controlo de uma rede de painéis digitais (*digital signage*).

Palavras-chave: interface do utilizador, aplicações móveis, desenvolvimento web, design, usabilidade.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Atualmente, os *smartphones* estão a mudar a forma como as pessoas interagem entre si e com o mundo. Pode até dizer-se que se criou uma certa dependência desses aparelhos "mágicos". Isto tudo não pelo aparelho em si, mas sim pelo que se consegue fazer com ele devido às inúmeras aplicações atualmente disponíveis.

Foi em 2007 e 2008 que tiveram lugar dois acontecimentos que marcaram o início desta área: foram o lançamento do iPhone pela *Apple* e do sistema operativo Android pela *Google*. Desde então, o desenvolvimento de aplicações para *smartphones* tem ganho cada vez mais adeptos, formando-se até grandes comunidades de partilha de ideias e de entre-ajuda. Apesar de ter apenas 5 anos, o desenvolvimento de aplicações para *smartphones* é uma área tecnológica já bastante desenvolvida. Mas, desenvolver para *smartphones* requer uma estratégia completamente nova em relação ao desenvolvimento para *desktops*. Surgiram novos desafios devido às particulariedades dos dispositivos móveis, cujas características são muito diferentes das de um computador normal. Existem pontos chave que precisam de ser bem compreendidos e tidos em conta para que as aplicações desenvolvidas para *smartphones* sejam eficientes. É preciso que se criem *guidelines* que confiram uma certa coerência nesta área tão propícia à fragmentação. Talvez o maior desafio se prenda com a interface, devido às suas dimensões reduzidas.

Apesar de tudo, a comunidade que desenvolve aplicações para *smartphones*, tem conseguido superar a maior parte dos desafios. Prova disso é a popularidade dessas mesmas aplicações. No entanto, há ainda um grande problema que é o facto do mercado dos *smartphones* estar segmentado em termos de plataformas (iOS, Android, Windows Phone 7, etc) utilizadas. Esta diversificação requer aos programadores um esforço adicional para portar as suas aplicações para as várias plataformas, se quiserem atingir um público-alvo o mais abrangente possível. Para colmatar esta e outras dificuldades, começou a ganhar relevância o desenvolvimento de aplicações para *smartphones* baseadas em tecnologias web. Com esta alternativa às aplica-

ções nativas, consegue-se atingir todo o público uma vez que a aplicação corre em qualquer *browser* recente instalado nos *smartphones*. Por outro lado, o HTML5 veio impulsionar esta nova vertente devido às suas novas funcionalidades, apesar de ainda não estar completamente implementado. Esta nova vertente está ainda nos seus primeiros passos, mas o facto é que já se encontram aplicações web de empresas de referência, como o facebook, google, sapo, público, etc.

1.1 Contextualização e Motivação

O projeto desenvolvido está inserido num contexto empresarial, uma vez que foi proposto pela empresa Ubisign[1]. *"A Ubisign é uma empresa de capitais próprios fundada em 2005 com o focus no desenvolvimento de soluções de topo para projetos profissionais de Digital Signage (painéis digitais). A tecnologia Ubisign facilita a convergência entre media digital, marketing, informação/entretenimento e dispositivos móveis permitindo a criação de soluções de valor acrescentado nos espaços de negócio. Estes espaços (pontos de venda, hotéis, salas de espera, etc.) oferecem aos seus clientes um conjunto inovador de serviços interactivos permitindo-lhes procurar, seleccionar, visualizar e publicar conteúdos digitais criando uma experiência muito mais rica com a marca"*[Cor11].

O uso de painéis digitais (*digital signage*) em lugares públicos, para fins informativos e comerciais, está a ter uma adopção crescente, nomeadamente no comércio, nas indústrias, nos hospitais e serviços públicos. Substituem os meios informativos tradicionais em papel e permitem apresentar conteúdo de uma forma dinâmica, conteúdo esse que pode ser publicidade, informações relevantes, vídeos de entretenimento, etc. Portanto, surgiu a necessidade de se desenvolver software de gestão e controlo desses mesmos painéis, sendo este o segmento de mercado em que a Ubisign desenvolve as suas soluções.

Uma rede de *digital signage* é constituída por um ou mais *players* (painéis) que reproduzem um determinado canal configurado pelo gestor da rede. Neste contexto, um canal corresponde à especificação do conteúdo (ficheiros multimédia, feeds RSS, previsão do tempo, etc.) e respetivo escalonamento. Tipicamente, os *players* são concebidos para executarem com uma disponibilidade 24/7. Sendo a elevada disponibilidade um requisito crítico nas redes de *digital signage*, é importante fornecer aos gestores/administradores da rede mecanismos de monitorização e controlo dos *players*, incluindo o suporte a cenários de mobilidade. Este contexto empresarial, aliado ao facto da necessidade de desenvolver uma aplicação para monitorizar e controlar uma rede de *digital signage* a partir de um dispositivo móvel, motivaram a realização do presente projeto, numa perspetiva de se obter um produto final funcional e útil.

[1] http://www.ubisign.com

1.2 Objetivos

São quatro os objetivos do trabalho conducente a esta dissertação. Um dos objetivos fundamentais é a análise das tecnologias base/subjacentes a esta área de desenvolvimento, tais como o HTML5, CSS3 e JavaScript. Outro é a identificação de boas práticas no desenvolvimento de aplicações web para *smartphones*, com especial atenção para a modelação e adaptação da interface. Esta última parte é especialmente importante uma vez que neste tipo de aplicações a interface tem um papel crítico devido às suas dimensões reduzidas, sendo necessário garantir uma aplicação intuitiva e com boa usabilidade.

Contudo o objetivo principal é definir uma abordagem para o desenvolvimento de aplicações web que forneçam uma experiência de utilização semelhantes às aplicações nativas, tendo como motivação o desenvolvimento de uma aplicação web destinada a monitorizar e controlar uma rede de *digital signage* a partir de um dispositivo móvel (e.g. telemóvel, *smartphone*, etc). Desta forma espera-se aplicar os conhecimentos e boas práticas adquiridas no estudo desta área. O que se pretende como produto final é uma aplicação que cumpra com os requisitos funcionais, mas principalmente que esteja bem adaptada para ser visualizada a partir de um *smartphone*. Isto significa cumprir com as boas práticas de desenvolvimento não só ao nível do código (separação entre visualização e lógica aplicacional), mas também em relação à modelação da interface.

A aplicação a desenvolver será baseada numa infra-estrutura de serviços a fornecer pela Ubisign, que suportará toda a lógica de negócio e de acesso aos dados necessária à concretização das funcionalidades requeridas. Portanto, o desenvolvimento da aplicação envolverá essencialmente o aspecto visual e a camada de acesso aos serviços.

A aplicação deverá implementar as seguintes funcionalidades:

- Autenticação no sistema da Ubisign.
- Listagem da informação sobre os *players* (estado de execução, data/hora do último check-in, canal sintonizado, estado da actualização do canal, etc.).
- Consulta de eventos de execução reportados por um determinado *player*.
- Alteração da sintonização de canal para um determinado *player*.
- Apresentação de gráficos/estatísticas sobre o estado geral da rede de *players*.

Para além de cumprir as boas práticas de desenvolvimento, a aplicação terá ainda como requisito adicional - e fundamental para a sua relevância - a capacidade de executar correctamente em *browsers* presentes nos sistemas operativos com mais representação no mercado.

Por fim, outro objetivo importante é fazer-se um levantamento de *frameworks* que existam disponíveis, cujo papel seja o de agilizar o processo de desenvolvimento de uma aplicação web para *smartphones*.

1.3 Estrutura da dissertação

Esta dissertação tem cinco capítulos principais, sendo eles o *Estado da Arte* (capítulo 2), *Casos de estudo* (capítulo 3), *Concepção e análise do problema* (capítulo 4), *Implementação* (capítulo 5) e *Testes* (capítulo 6).

Após o capítulo introdutório (capítulo 1), são abordados no capítulo 2 vários temas que estão relacionados com o desenvolvimento e modelação de aplicações para *smartphones*. Começa-se por uma análise a duas abordagens distintas no desenvolvimento de aplicações para *smartphones*. Mais concretamente, são analisadas e comparadas aplicações nativas com aplicações web. A secção seguinte está reservada para a análise do HTML5, que tem um papel fundamental no desenvolvimento desta área, como veremos. Depois, são exploradas diversas *frameworks* de desenvolvimento, que simulam a interface e as interações com esta, como se se tratasse de uma aplicação nativa. A última secção do capítulo, mas não menos importante, é dedicada à análise das boas práticas no desenvolvimento de aplicações em geral para *smartphones*. São descritos os aspetos fundamentais relacionados com a interface, para que esta seja intuitiva e de simples utilização.

No capítulo 3 são analisados dois websites que se enquadram no contexto do tema desta dissertação. Mais concretamente, analisam-se duas adaptações para *smartphones* de websites complexos e, portanto, não otimizados para serem visualizados a partir de pequenos dispositivos móveis.

Os três capítulos seguintes (4, 5 e 6) são dedicados exclusivamente ao projeto que serviu de base a esta dissertação de mestrado e que motivou o estudo nesta área. No capítulo 4 é feita uma análise ao problema que foi proposto, onde se aborda a concepção da aplicação desenvolvida. No capítulo 5, é descrita de uma forma pormenorizada toda a implementação da aplicação. No capítulo 6, são descritos os vários tipos de teste a que a aplicação foi sujeita.

Finalmente, no capítulo 7, são expostas as conclusões. Apresenta-se uma visão crítica da área em questão, onde são mencionados aspetos importantes que precisam de mudar. Faz-se também uma reflexão em relação ao projeto desenvolvido, com um balanço final sobre o desenvolvimento da aplicação baseada em tecnologias web. Algumas ideias relativamente ao futuro desta área são também afloradas.

Capítulo 2

Estado da arte

O desenvolvimento de aplicações móveis é uma área bastante recente e que tem tido um crescimento grande devido à popularidade e massificação dos *smartphones* ou mais genéricamente dispositivos móveis. A evolução destes dispositivos tem sido grande, o que impulsiona ainda mais esta área emergente.

Neste capítulo vamos abordar vários temas relacionados com o desenvolvimento e modelação de aplicações móveis, para além de alguns temas mais genéricos, mas não menos importantes. Vamos começar por analisar duas abordagens distintas: o desenvolvimento de aplicações nativas e aplicações web. Na secção seguinte vamos abordar as tecnologias consideradas fundamentais para o desenvolvimento de aplicações web, o HTML5 e CSS3. Depois são analisadas e comparadas diversas *frameworks*, e por fim, na última secção são apresentadas algumas *guidelines* para o desenvolvimento de aplicações web para dispositivos móveis.

2.1 Aplicações nativas vs. aplicações web

Nesta área, um tema que tem merecido alguma atenção são as diferenças entre desenvolver uma aplicação nativa e desenvolver uma aplicação web, uma vez que são duas abordagens bem distintas. Impõe-se, portanto, saber qual delas é a melhor estratégia para desenvolver uma aplicação, tendo em conta todos os seus requisitos e público alvo. Por um lado, temos uma abordagem mais sólida e antiga que é fazer aplicações que corram nativamente no sistema operativo dos dispositivos, por outro temos o desenvolvimento de aplicações baseadas em linguagens web, mais especificamente em HTML5 e JavaScript. Este tema tem a particularidade de ter vindo a sofrer algumas mudanças de paradigma em relativamente pouco tempo. Um fator a ter em conta para esta instabilidade é a evolução das tecnologias subjacentes a cada tipo de aplicações. Não só o *hardware* dos dispositivos móveis tem vindo a evoluir consideravelmente nestes últimos 3 a 4 anos, mas também

se tem assistido a um desenvolvimento das tecnologias e *standards* da web, nomeadamente o HTML5 e, embora com menos impacto, o CSS3 [VN10].

Uma aplicação nativa é uma aplicação que é desenvolvida para uma determinada plataforma ou sistema operativo, tal como o Android, iOS, Windows Phone, etc. É uma aplicação que corre nativamente no dispositivo, tendo por isso que ser transferida e instalada para poder ser utilizada. Por sua vez, uma aplicação web é simplesmente uma página web que pode ser visualizada num *browser*. Sendo utilizadas a partir do *browser* do dispositivo, não é necessário instalar nada para além deste.

Existem muitos fatores que podem ajudar a decidir entre desenvolver uma aplicação nativa ou uma aplicação web, sendo que uns podem ter um maior peso na decisão do que outros. Nesta secção vamos explorar em detalhe as vantagens e desvantagens de cada uma destas abordagens, não só do ponto de vista dos programadores como também dos utilizadores.

2.1.1 Experiência do utilizador

No geral, as aplicações nativas são mais rápidas do que as aplicações web, oferecendo também uma experiência de utilização mais sólida e integrada pois foram desenvolvidas tendo em conta as especificações dos dispositivos alvo. A vantagem das aplicações nativas serem mais rápidas é um fator decisivo quando se está a desenvolver uma aplicação que tenha que ser fluída e com tempos de resposta muito rápidos. Um género de aplicações que se enquadram neste cenário são os jogos. As aplicações nativas, como tipicamente seguem as convenções e normas das plataformas destino, ficam com uma interface mais intuitiva e apelativa para os utilizadores, visto já estarem familiarizados com esse tipo de interfaces e o modo de interagir com elas. Relativamente às interfaces, numa aplicação nativa é possível ter total controlo sobre o seu *layout* bem como usar a aplicação em modo *full-screen*, enquanto que nas aplicações web os programadores devem ter em consideração o modo como a aplicação é visualizada nos diferentes *browsers*[CL11].

2.1.2 Acesso às funcionalidades do dispositivo

Se a aplicação precisar de ter acesso às várias funcionalidades do dispositivo, tal como a câmara, GPS, acelerómetro, bússula, lista de contactos, etc, então a melhor opção neste momento é uma aplicação nativa pois têm acesso sem restrições a todas essas funcionalidades. Embora neste campo as aplicações nativas ainda sejam as favoritas, a tendência é que esta vantagem seja atenuada devido ao aumento do número de funcionalidades a que uma aplicação web tem acesso. Neste momento já existe uma API de geolocalização, fazendo com que uma aplicação web tenha acesso à localização geográfica do dispositivo de uma forma semelhante à que uma aplicação nativa teria [W3C12b]. Quanto às restantes funcionalidades dos dispositivos, atualmente

a *World Wide Web Consortium* (W3C) está a trabalhar num conjunto de API's que dariam acesso às funcionalidades mais importantes presentes nos dispositivos móveis. É, portanto, uma questão de tempo até que o acesso às funcionalidades do dispositivo deixe de ser um fator decisivo.

2.1.3 Modelos de negócio

Do ponto de vista comercial e no sentido de estabelecer um modelo de negócio viável e lucrativo, existem algumas diferenças entre estas duas abordagens.

As aplicaçõs nativas, quando disponibilizadas pelos meios ou lojas oficiais (Ex.:*App Store, Android market*), são facilmente comercializadas. Esta via de comercialização é muito atrativa para os autores das aplicações pois, desta forma, não se têm que preocupar com toda a logística associada à venda de um produto e por outro lado conseguem atingir um público alvo muito alargado. Contudo, como este é um mercado onde os preços são relativamente baratos e onde existe uma oferta excessiva, existem muitas aplicações que não conseguem ser comercializadas com sucesso devido à falta de visibilidade [HO11].

Quanto às aplicações web, os mecanismos para a sua comercialização têm que ser implementados e portanto isto requer um esforço de desenvolvimento adicional. Existem várias formas de tornar uma aplicação web comercialmente atrativa como por exemplo implementar um sistema de registos pagos, onde só pessoas que tenham pago possam aceder à aplicação [Cle09].

Uma forma muito comum de obter algum retorno monetário e que pode ser usado nas duas abordagens é o uso de publicidade. Geralmente o uso de publicidade só é utilizado em aplicações que à partida são gratuitas e é comum haver uma versão paga onde não existe publicidade [DV11].

Não se pode, portanto, à partida dizer qual delas é melhor porque também vai depender muito do tipo de aplicação e do público alvo. Ambas as abordagens permitem, embora de maneiras diferentes, que a aplicação possa ser comercialmente viável no sentido de obter lucro com a sua comercialização.

2.1.4 Facilidade na distribuição/*deployment*

Quanto à facilidade com que se lançam ou disponibilizam as aplicações no mercado existem algumas diferenças significativas nas duas abordagens. Por um lado, com as aplicações nativas, normalmente apenas temos a hipótese de as disponibilizar pela loja oficial. Por outro lado, com as aplicações web, não estamos dependentes de entidades terceiras na medida em que a aplicação é disponibilizada como se se tratasse de uma página web. Ao estarmos dependentes das lojas oficiais para disponibilizar-mos as nossas aplicações, ficamos sujeitos às regras e barreiras impostas pelas mesmas, o que por si só pode

ser um fator impeditivo. A manutenção de uma aplicação é também um pormenor importante a ter em conta no lançamento de uma aplicação. O facto dos utilizadores muitas das vezes não atualizarem as aplicações instaladas, faz com que seja mais atrativo desenvolver e disponibilizar as aplicações via web, uma vez que desta forma se garante que os utilizadores utilizam sempre a última versão. Em suma, neste aspeto em concreto o prato da balança é favorável às aplicações web.

2.1.5 Desempenho

Sem dúvida alguma que o desempenho é um aspeto crítico e tem um papel fundamental na experiência de utilização de qualquer aplicação. Uma aplicação pouco otimizada na execução de certas funções ou com muita latência na sua interação pode ser à partida excluída pelo simples facto de não proporcionar uma boa experiência de utilização [GBD00]. De certa forma o desempenho é um campo subjetivo pois existem muitas variáveis que podem interferir no funcionamento de uma aplicação, nomeadamente as características do hardware, a quantidade de memória livre disponível, a estabilidade e largura de banda da conexão à internet, etc. Existem várias métricas para quantificar o desempenho, sendo as mais comuns a latência e o tempo de execução que uma operação ou função demora[CL11].

A latência é um parâmetro especialmente importante para as aplicações web e para todas as aplicações que de certa forma recebem e enviam informação pela internet. Além disso, a latência também se manifesta de outras formas, como por exemplo no tempo que demora uma aplicação a ser carregada para a memória ou no caso do JavaScript há uma latência associada à interpretação/*parsing* do código[Mul11].

No que diz respeito à latência, as aplicações nativas têm uma vantagem considerável. Por um lado, uma aplicação nativa já está presente no dispositivo e portanto não será necessário transferi-la, o que já não é verdade no caso das aplicações web. Por outro lado, a fluidez de uma interface nativa é maior quando comparada com a interface de uma aplicação web, sendo que esta última está de certa forma dependente do *browser* utilizado. Depois, quanto ao tempo de execução de determinadas tarefas, as aplicações nativas têm outra vez vantagem face às aplicações web. Esta vantagem é devido ao código nativo ser mais eficiente do que código JavaScript, embora não haja uma diferença muito grande[FLWZ07].

Portanto, de uma forma geral em termos de desempenho, as aplicações nativas têm uma indiscutível vantagem face às aplicações web.

2.1.6 Conclusão/Veredicto

Na tabela 2.1 encontram-se os critérios de avaliação atrás mencionados, de uma forma resumida, dando uma visão geral dos pontos positivos e ne-

	Aplicações web	Aplicações nativas
Experiência do utilizador	-	+
Acesso às funcionalidades do dispositivo	-	+
Modelos de negócio	+	+
Facilidade na distribuição (*deployment*)	+	-
Desempenho	-	+

Tabela 2.1: *Resumo das vantagens e desvantagens de cada uma das abordagens.*

gativos de cada uma das abordagens. Dadas as vantagens e desvantagens de cada uma delas, cabe às empresas ou organizações avaliar qual delas é a melhor no contexto de negócio em que as aplicações a desenvolver se inserem. Em alguns casos até ambas as abordagens são válidas e fazem sentido. Como conclusão, nenhuma das abordagens é perfeita para todas as situações e actualmente tanto aplicações nativas como aplicações web devem ser igualmente consideradas.

2.2 HTML5

O HTML (HyperText Markup Language) é uma linguagem de marcação utilizada para estruturar e representar conteúdo sob a forma de uma página web. Neste caso em particular, o HTML5 é a iteração mais recente da linguagem HTML, que sofreu muitas das modificações necessárias para fazer face às necessidades atuais do mundo web. O HTML5 deriva da cooperação entre duas organizações, a World Wide Web Consortium (W3C) e a Web Hypertext Application Technology Working Group (WHATWG). E vem substituir a anterior especificação HTML4.01, que já contava com uma década de existência desde que foi publicada pela W3C em 1999.[VN10]

2.2.1 Novas funcionalidades do HTML5

O HTML5 está a ser desenvolvido com vários objetivos em mente, nomeadamente substituir os *standards* de video e audio proprietários por *standards* abertos, permitir que aplicações web se comportem de uma forma semelhante às aplicações nativas, permitir serviços baseados na geolocalização, e tornar a sintaxe do HTML mais organizada.[Hoy11]

Uma das funcionalidades mais esperadas é o suporte nativo da reprodução de audio e vídeo pelo *browser*. Atualmente os *browsers* estão dependentes de *plugins* de terceiros para esse efeito, como por exemplo: *Adobe Flash, Microsoft Silverlight, Apple Quicktime*. O uso destes plugins é praticamente transparente quando um utilizador está num portátil ou desktop, mas já se torna problemático quando se está a usar outro tipo de dispositivos, no-

meadamente *smartphones*, devido à falta de plugins compatíveis. O HTML5 vem colmatar esta falha permitindo que se usem as tags <video> e <audio>, sendo depois os próprios *browsers* que nativamente fazem a reprodução multimédia desse conteúdo[LS11]. Embora ainda não tenha sido especificado o formato desses conteúdos multimédia, o mais provável é serem o *Ogg Theora* e o *H.264* que são *standards* abertos muito usados. Atualmente existem algumas divergências quanto ao formato a suportar. A *Microsoft* e a *Apple*, com o Internet Explorer e Safari, respetivamente, estão a pensar suportar o formato H.264, enquanto que o grupo Mozilla Firefox e o grupo do *browser* Opera suportam o Ogg Theora. O *browser* da *Google*, o Chrome, irá suportar ambos os formatos.

Haverá também suporte a bases de dados SQL no lado do cliente, tal como *offline caching* de ficheiros. Para os utilizadores de *smartphones*, o suporte a *offline caching* vai ter especial importância pois nem sempre têm conectividade à rede. Uma vantagem de se poderem guardar ficheiros, tal como imagens, código, dados do utilizador, etc, é o tempo de carregamento e latência da aplicação ser menor, minimizando-se também o tráfego [LS11].

De entre os novos elementos introduzidos pelo HTML5, o *Canvas* é certamente um dos mais importantes. É um elemento que consiste numa região com uma determinada altura e largura onde é possível renderizar gráficos vectoriais ou outro tipo de imagens. Este elemento é controlado através de *JavaScript*, permitindo também a interação do utilizador através do rato e teclado. É portanto, um elemento que pode conter não só figuras geométricas simples, mas também jogos ou animações complexas de imagens [FF11].

O HTML5 também irá introduzir novas funcionalidades relacionadas com a geolocalização. Basicamente vai ser possível uma aplicação web ter acesso à localização geográfica do utilizador caso este esteja num aparelho com GPS. É, portanto, uma funcionalidade direcionada principalmente para os *smartphones* e outros dispositivos móveis. Isto vai fazer com que seja possível criar aplicações mais dinâmicas e que tirem partido da localização do utilizador para, por exemplo, fornecer informação priveligiada de eventos ou locais na sua proximidade. São, portanto, inúmeras as aplicações que podem tirar partido desta funcionalidade. As redes sociais podem também tirar partido desta funcionalidade para, por exemplo, informar um grupo específico da sua rede de contactos da sua localização e vice-versa. Temos também o caso da publicidade, que poderá utilizar a informação da localização do utilizador, para publicitar lojas, eventos ou restaurantes que estejam na sua proximidade. Existe, no entanto, uma preocupação geral e legítima no que concerne à privacidade dos utilizadores, nomeadamente o uso indevido deste tipo de informação. É certo que é o utilizador que tem que explicitamente dar à aplicação o acesso a esta funcionalidade, mas depois não tem controlo sobre como é usada a informação relativa à sua geolocalização [Hol11][W3C12b].

Ao nível da sintaxe também surgirão algumas mudanças de modo a ficar melhor estruturada e mais limpa. Uma grande alteração está relacionada

com a presença de estilos nos elementos. Na versão anterior do HTML os estilos presentes nos próprios elementos tinha precedência em relação aos estilos contidos nas folhas de estilo (*Style Sheets*). Nesta nova versão os estilos definidos nas folhas de estilo terão precedência. Esta mudança é totalmente compreensível e vai de encontro às boas práticas no desenvolvimento de páginas web, que é separar o conteúdo da forma de apresentação ou estilo. Para além disto, ixistirão novas tags, como por exemplo, *<article>*, *<aside>*, *<nav>*, *<footer>*, *<summary>*, etc permitindo estruturar melhor o conteúdo em blocos numa página web.[W3C12a]

2.2.2 Vantagens do HTML5

Talvez a maior vantagem do HTML5 é o facto de poder vir a ser suportado por praticamente todas as plataformas existentes, quando a sua norma estiver completa e os *browsers* a implementarem, sendo por isso considerado multi-plataforma. Os sistemas operativos dominantes, iOS, Android, BlackBerry, Windows Phone, diferem significativemente nas tecnologias de desenvolvimento das respectivas aplicações nativas. As aplicações para iOS são desenvolvidas em *Objective-C*, para Android são em Java, para BlackBerry usa-se Java ou JavaScript e no caso do Windows Phone as aplicações são desenvolvidas tendo por base a framework .Net usando C#. Esta grande heterogeneidade implica à partida que aplicações nativas não possam ser multi-plataforma, fazendo com que tenha que ser feito um esforço para se adaptar uma aplicação às restantes plataformas. Esta questão da diversidade de plataformas é ultrapassada desenvolvendo as aplicações em HTML5 e JavaScript, podendo assim as empresas que desenvolvem aplicações apenas concentrarem os seus esforços e recursos numa plataforma de desenvolvimento comum à maioria dos dispositivos.

Quase todos os sistemas operativos suportam vários dispositivos com características bastante diferentes, como o tamanho do ecrã, a sua resolução, se é *touch screen* ou não, a densidade do ecrã, etc. Cabe portanto aos programadores terem a responsabilidade de garantir que a suas aplicações vão ser corretamente visualizadas nos vários dispositivos. Este problema pode ser em grande parte resolvido usando as novas funcionalidades que o CSS3 veio introduzir, tais como as "media queries", as propriedades "text overflow" e "word wrap", como também a possibilidade de usar tamanhos relativos de elementos. Estas novas funcionalidades permitem um maior controlo e flexibilidade na forma como as páginas podem ser visualizadas nos vários dispositivos.

Outra vantagem relativamente ao desenvolvimento de aplicações em HTML5 e JavaScript é o facto de existirem muitas bibliotecas e frameworks de desenvolvimento que agilizam todo esse processo e algumas proporcionam até interfaces muito semelhantes às das aplicações nativas. Frameworks e bibliotecas como o jQtouch, jQuery Mobile e Sencha Touch são apenas alguns

exemplos de ferramentas que permitem aos programadores criar interfaces ricas e com uma usabilidade e comportamento semelhante às aplicações nativas. Tendo todas estas ferramentas ao dispor, o desenvolvimento de uma aplicação pode focar-se mais nos aspetos funcionais.

2.2.3 Conclusões

Apesar de haver um grande entusiasmo relativamente às funcionalidades e potencialidades do HTML5, existem ainda alguns obstáculos a ultrapassar. O maior obstáculo é o ritmo lento a que estão a ser especificados e implementados os novos *standards*. As organizações responsáveis pelo seu desenvolvimento, nomeadamente a *W3C*, já o estão a fazer há alguns anos.

Embora algumas funcionalidades já tenham sido implementadas em alguns *browsers*, o tempo de espera até que os *standards* sejam finalizados pode fazer com que algumas funcionalidades sejam abandonadas pelos programadores em detrimento de outras, criando assim alguns problemas no futuro relacionados com a fragmentação das funcionalidades suportadas.

O facto do HTML5 vir redefinir o conceito de aplicações web, tornando-as mais próximas das aplicações nativas origina vários problemas de segurança, decorrentes do facto de ser possível armazenar e correr localmente dados e código de aplicações web. Ao contrário, as versões anteriores do HTML apenas permitiam que fossem armazenados localmente pequenos ficheiros de texto (*Cookies*), diminuindo assim potenciais falhas de segurança.

2.3 Frameworks para desenvolvimento de aplicações web

"A framework is a set of cooperating classes that make up a reusable design for a specific class of software. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsabilities and collaborations. A developer customizes the framework to a particular application by subclassing and composing instances of framework classes." [GHJV95]

Nesta secção vamos explorar diversas frameworks para o desenvolvimento de aplicações web para dispositivos móveis. Estas frameworks são na sua generalidade bibliotecas ou APIs javascript que agilizam o processo de desenvolvimento e abstraem conjuntos de funcionalidades que são muitas vezes usadas, agilizando desta forma todo o processo de desenvolvimento. No caso particular das frameworks para o desenvolvimento de aplicações web móveis, estas incidem principalmente na interface da aplicação para que o utilizador tenha uma experiência de utilização semelhante às aplicações nativas, tendo assim as vantagens inerentes a esse tipo de interfaces.

Um facto curioso é que a grande parte das *frameworks* atuais tem como

base a interface *standard* das aplicações desenvolvidas para o iPhone. Isto acontece porque a interface das aplicações para o iPhone segue várias normas e padrões de usabilidade, tem também já alguma maturidade e é bastante popular devido à grande comunidade de *developers* e utilizadores. São interfaces muito intuitivas devido à validação a que foram sujeitas por parte da sua grande comunidade de utilizadores. Desde cedo que a política da *Apple* tem sido muito clara quanto à sua interface e quanto à sua usabilidade, disponibilizando para isso manuais e *user interfaces guidelines* que têm que ser minimamente cumpridas para as aplicações serem aceites. Ora isto levou a que rapidamente se formasse um modelo de desenvolvimento e esteriotipagem da interface e modelos de interação com esta. É portanto um modelo de interface adequada aos dispositivos móveis, tirando partido da grande experiência da comunidade de desenvolvimento de aplicações móveis nativas.

No caso do *Android* encontramos um sistema mais heterogéneo, embora também existam padrões e *guidelines* para se desenhar e desenvolver a interface. Mas neste caso não são tão pormenorizadas nem passam pelo grande controlo de usabilidade que existe na *app store* da *Apple*. Como consequência não há uma grande uniformização no que diz respeito à interacção e implementação da interface.

De seguida são exploradas as *frameworks* com mais relevo em termos de funcionalidades e popularidade.

2.3.1 iUI

A framework iUI[1] foi inicialmente criada por Joe Hewitt e o seu objectivo principal era simplesmente transformar a interface de uma página web HTML, tornando-a numa interface rica, muito orientada à experiência do utilizador e que cumprisse os padrões que a própria *Apple* exige nas suas aplicações nativas. A versão inicial do iUI tinha o nome de "iphonenav" e era uma versão um pouco arcaica quando comparada com a versão atual. Ainda antes de ser oficialmente lançada com o nome iUI, já tinha tido uma boa recetividade pois tinha sido já adotada em muitos projetos.[Hew07]

Esta *framework* tira partido das tecnologias Javascript, CSS e HTML5 para tornar as interfaces web o mais parecidas com as aplicações nativas que se encontram nos dispositivos da Apple, tal como o iPhone, iPod e o iPad. Mas a forma como esta *framework* atinge os seus objectivos é de certa forma diferente daquilo que se encontra noutras *frameworks* similares. Aquilo que o iUI faz é extender o HTML standard, mapeando algumas tags HTML em comportamentos específicos da interface. Por exemplo os links de uma página (tag <a>) ficam com uma animação associada para que quando forem carregados ocorra um efeito de deslizamento antes de se visualizar a página

[1] http://www.joehewitt.com/blog/introducing_iui.php

do link. Portanto, todos os links são automaticamente modificados de forma a usarem AJAX (*Asynchronous JavaScript and XML*) para carregar páginas de forma dinâmica, tornando desta forma a navegação dentro da aplicação bastante semelhante às aplicações nativas.

Um problema bastante comum nas aplicações web móveis é a apresentação de listas bastante longas de forma eficiente. Com o iUI, estas listas podem ser carregadas incrementalmente, fazendo com que a lista não seja inicialmente toda carregada para a página. O modo como isto é conseguido é bastante simples, basta colocar no fim duma lista um link com o campo *target="_ replace"* e o que o iUI faz é carregar esse URL via AJAX e depois vai substituí-lo pelos conteúdos carregados. Em vez de se usar um link em que o utilizador tem que o clicar para carregar mais uma porção da lista, pode-se usar javascript para detectar quando o utilizador chegar com o *scroll* ao fim da lista para depois carregar automaticamente outra porção. A submissão de formulários em páginas também é tratada pelo iUI porque normalmente a submissão de um formulário vai-nos direccionar para outra página, o que faz com que ao nível da interface a transição não seja suave ao ponto de se poder comparar a uma aplicação nativa. O que o iUI faz é submeter os forms via AJAX e depois o resultado é apresentado na página dinamicamente. Estas são apenas algumas das funcionalidades implementadas pelo iUI.[Hew07][Gil12]

Esta abordagem bastante específica da *framework* iUI permite que o programador não tenha que utilizar javascript para desenvolver a sua plicação no que toca à interface. No entanto é uma *framework* que tem funcionalidades simples, pelo que só é aconselhável para o desenvolvimento de pequenos e médios projetos. Quanto ao seu desempenho, não é das melhores *frameworks*, mas mesmo assim é aceitável na maioria dos dispositivos.[Hew07]

2.3.2 jQTouch

O jQTouch[2] é um plugin jQuery para o desenvolvimento de aplicações web para dispositivos móveis como o iPhone, iPod Touch ou qualquer outro *smartphone* atual [Kan10].

Foi desenvolvido e adaptado para os *browsers* que têm como motor gráfico o WebKit, o que na prática abrange praticamente todo o segmento de *browsers* que vêm por omissão nos dispositivos móveis. No entanto, se for usado num bowser que não tenha o motor gráfico WebKit, também funciona tudo à excepção das animações e transições entre páginas. Isto acontece porque estas são feitas nativamente pelo WebKit, o que é uma grande vantagem uma vez que ficam mais fluídas e mais leves [Kan12a].

Logo à partida esta *framework* tem a vantagem de usar jQuery, uma biblioteca javascript bastante conhecida e utilizada, o que faz com que a

[2] http://jqtouch.com

curva inicial de aprendizagem desta framework seja boa. Depois, quanto ao processo de aprendizagem e uso da *framework*, é relativamente simples existindo bastante documentação de suporte na sua página oficial. Quanto às funcionalidades e outros aspetos da *framework*, pode dizer-se que é bastante completa a quase todos os níveis. É bastante configurável no que diz respeito ao *design*, através de temas flexíveis. Pode-se também adicionar ou extender funcionalidades através do jQuery. Outra característica interessante é o mecanismo de eventos, que deteta determinadas ações do utilizador e lança os respetivos eventos que depois são tratados pelas funções respectivas. Por exemplo, podemos subscrever eventos tais como "a orientação do dispositivo mudou" ou "o utilizador fez um movimento de *swipe* no ecrã", e depois tratamos esses eventos de forma adequada [Kan12b]. O jQtouch tira partido de algumas funcionalidades do HTML5, nomeadamente o acesso à geolocalização e a possibilidade da aplicação ficar em cache no dispositivo de modo a funcionar *offline*[Kan12c].

Existem também algumas funcionalidades que não estão presentes nesta *framework*, funcionalidades essas que na maioria dos casos só são necessárias para desenvolver aplicações complexas. Tratam-se de funcionalidades ao nível do dispositivo tal como o acesso ao acelerómetro (é possível apenas detetar alterações na sua orientação), à câmara fotográfica, aos contactos, sistema de notificações, gestão local de ficheiros, etc. A única funcionalidade ao nível do dispositivo que está presente é, como já foi referido, o acesso à geolocalização a partir da API do HTML5.

Em comparação com a *framework* iUI, esta é mais avançada e complexa sendo portanto mais direcionada a programadores JavaScript. Tem também a vantagem de estar mais otimizada e portanto apresenta uma interface com maior rapidez e fluidez de resposta [Hei10].

Importa aqui referir, que esta *framework* foi utilizada no desenvolvimento do projeto, uma vez que pelas suas funcionalidades e características se adequava ao que era necessário.

2.3.3 PhoneGap

PhoneGap[3] é uma *framework* de desenvolvimento *open-source*, que permite aos programadores desenvolverem as suas aplicações usando tecnologias web tal como o HTML5, Javascript e CSS3 e depois distribuírem-nas como aplicações nativas[Pho12a]. Na figura 2.1 está ilustrado o conceito geral desta *framework*.

Esta *framework* usa portanto uma abordagem diferente das que vimos anteriormente. As aplicações são desenvolvidas como se se tratasse de uma aplicação web usando HTML5, JavaScript e CSS3. Depois são compiladas através do PhoneGap e o resultado final é uma aplicação nativa que irá correr

[3] http://www.phonegap.com

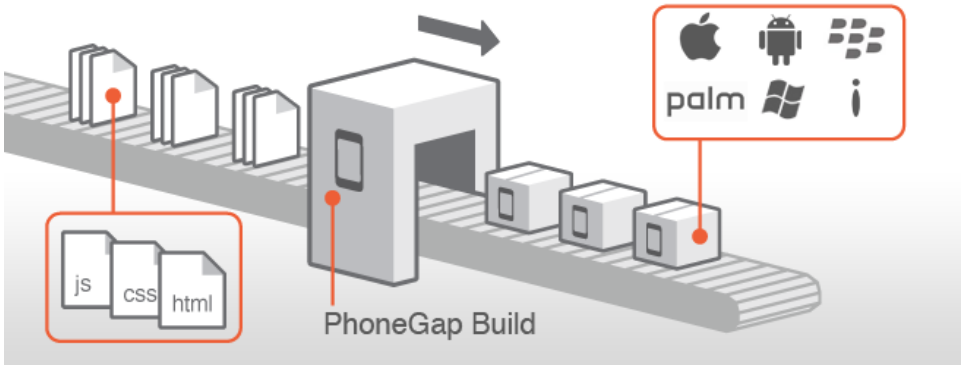


Figura 2.1: *Processo de desenvolvimento com a framework PhoneGap[Pho12b]*

no dispositivo. Portanto pode-se dizer que o PhoneGap atua como um *wrapper* de uma página web e que disponibiliza a esta o acesso a funcionalidades do dispositivo através de API's.[Pho12b]

O PhoneGap suporta os seguintes sistemas/plataformas:

- Android
- iPhone
- Blackberry
- Symbian
- Windows Phone 7
- WebOs
- Bada

Na tabela 2.2 podemos ver quais as funcionalidades nativas disponibilizadas na API do PhoneGap e em que sistemas operativos ou plataformas elas são suportadas. Podemos ver que nos 3 sistemas mais conhecidos e usados (iOS, Android e Windows Phone 7) o acesso às funcionalidades nativas é completo. Nos restantes sistemas, o acesso a algumas funcionalidades ainda não é suportado por diversas razões, mas poderá vir a sê-lo em próximas versões.

Existem muitas vantagens que levam as empresas e programadores a optarem por usar uma *framework* como o PhoneGap em detrimento de desenvolver as aplicações usando as linguagens e SDK's nativos. De seguida são enumeradas as vantagens mais relevantes de usar a *framework* PhoneGap. Em primeiro lugar, é bastante provável que numa empresa ou equipa de desenvolvimento já haja um conhecimento aprofundado das tecnologias web (HTML e JavaScript) ao contrário das tecnologias e linguagens nativas dos vários sistemas. Temos, por exemplo, o caso do desenvolvimento

Feature	iOS	Android	WebOS	WP7	Symbian	Bada
Accelerometer	Yes	Yes	Yes	Yes	Yes	Yes
Camera	Yes	Yes	Yes	Yes	Yes	Yes
Compass	Yes	Yes	No	Yes	No	Yes
Contacts	Yes	Yes	No	Yes	Yes	Yes
File	Yes	Yes	No	Yes	No	No
Geolocation	Yes	Yes	Yes	Yes	Yes	Yes
Media	Yes	Yes	No	Yes	No	No
Network	Yes	Yes	Yes	Yes	Yes	Yes
Notification(alert)	Yes	Yes	Yes	Yes	Yes	Yes
Notification(sound)	Yes	Yes	Yes	Yes	Yes	Yes
Notification(vibration)	Yes	Yes	Yes	Yes	Yes	Yes
Storage	Yes	Yes	Yes	Yes	Yes	No

Tabela 2.2: *Funcionalidades nativas suportadas pela framework PhoneGap[Pho12c]*

para a plataforma iOS (iPhone, iPod, iPad), que requer conhecimentos em Objective-C, que é uma linguagem pouco usada e conhecida. Portanto, o conhecimento prévio de HTML e JavaScript aliado ao facto de que com essas linguagens se consegue desenvolver uma aplicação que funciona nas várias plataformas existentes faz com que seja mais eficiente concentrar esforços e desenvolver aplicações usando esta abordagem. A segunda vantagem está de certa forma ligada com a primeira, e é a possibilidade de se usarem *toolkits* ou *frameworks* JavaScript já existentes tal como jQuery Mobile, Sencha Touch, Dojox Mobile, etc. Esta modularidade e o facto de haver inúmeras bibliotecas JavaScript torna esta abordagem de desenvolvimento bastante atraente. Um dos pontos fortes desta *framework* são as API's que dão acesso às funcionalidades principais dos dispositivos. Outra vantagem prende-se com própria arquitectura do PhoneGap que possibilita a extensão de funcionalidades para que uma aplicação tenha acesso a mais funcionalidades nativas do que aquelas que estão presentes nas API's atuais da *framework*.[Pho11]

Existem, também, algumas limitações nesta *framework* que devem ser tidas em consideração. A que tem mais impacto é o facto das aplicações desenvolvidas usando esta abordagem serem muito mais lentas e menos otimizadas do que as aplicações nativas "normais". Poderão, portanto, surgir problemas de desempenho em aplicações complexas ou em dispositivos mais

antigos. Para desenvolver jogos complexos ou trabalhar com gráficos intensivos a melhor solução é usar openGL em conjunto com a linguagem de programação nativa em vez de usar esta *framework*. Outro aspeto a ter em consideração é que nem todas as funcionalidades nativas são suportadas nas várias plataformas, o que faz com que uma aplicação possa funcionar bem num sistema e não funcionar noutro. No que diz respeito à interface, tem as limitações inerentes à programação web que faz com que seja difícil simular uma interface nativa.[Pho11]

2.3.4 Outras frameworks

Existem imensas *frameworks* para além das mencionadas, como por exemplo: iWebkit[4], Sencha Touch[5], Appcelerator[6], Sproutcore Touch[7], etc. Na generalidade todas estas frameworks são de alguma forma semelhantes, embora com níveis diferentes de maturidade e suporte de funcionalidades, sendo que todas elas visam o desenvolvimento de aplicações web com implementações ao nível da interface semelhante às aplicações nativas.

2.3.5 Benefícios que advêem do uso de frameworks

Com a utilização de *frameworks* obtemos vários benefícios ou vantagens tais como a modularidade, extensibilidade, reusabilidade e a inversão de controlo encontrado em muitas *frameworks*. As *frameworks* aumentam a modularidade das aplicações porque geralmente encapsulam os detalhes complexos da sua implementação fornecendo apenas interfaces ou pontos de extensão de uma forma organizada e simplificada. É também mais fácil manter e evoluir uma aplicação que foi desenvolvida usando como base uma *framework*, porque esta está melhor estruturada e compartimentalizada. Outro aspeto positivo é o facto das *frameworks* promoverem a reutilização e partilha do conhecimento, pois geralmente são a acumulação de conhecimento e experiência de programadores das mais diversas áreas. As interfaces que as *frameworks* disponibilizam são apenas pontos de extensão que são usados pelas várias aplicações, partilhando assim um conjunto de funcionalidades base. Desta forma, o desenvolvimento de uma aplicação não começa do zero, aumentando assim a produtividade e qualidade do produto final, visto que se começa a desenvolver partindo de uma base supostamente sólida e com bastante maturação. Aumenta também a interoperabilidade do software, pois várias instâncias da aplicação partilham características comuns. Oferecem pontos de extensão para que os programadores possam complementar as

[4] http://snippetspace.com/portfolio/iwebkit/
[5] http://http://www.sencha.com/products/touch
[6] http://www.appcelerator.com/
[7] http://sproutcore.com/

suas funcionalidades para assim fazer uma aplicação específica para um determinado contexto.[HF98]

Finalmente, algumas *frameworks* seguem um padrão de desenvolvimento chamado inversão de controlo (*Inversion of Control* ou *IoC*, em inglês). Inversão de controlo é também conhecida como princípio de Hollywood: "Don't call us, we will call you" [Lar04]. Inversão de controlo é um padrão de desenvolvimento caracterizado por uma inversão na sequência de chamadas dos métodos em relação ao que o programador em geral está habituado. Isto significa que este controlo não é feito directamente pelo programador, mas sim por uma infraestructuta de software (*container*) que toma o controlo sobre a sua execução. Um fluxo normal de execução num determinado programa caracteriza-se por aspetos como a criação de objetos, entidades ou o inicio e fim da execução estarem sobe o controlo do programador. A inversão do controlo ocorre quando, em vez de se criar explicitamente um código ou acompanhar todo o ciclo de vida de uma execução, o programador delega alguma dessas funcionalidades para um terceiro (Injeção de dependência) [FS97, Joh97].

2.3.6 Desafios ao usar frameworks

O uso de *frameworks* traz vantagens para toda a equipa de desenvolvimento reduzindo o esforço global e aumentando a qualidade final do produto. Contudo, traz também uma série de desafios e problemas que têm de ser resolvidos, sob pena de comprometerem todo o projeto. O primeiro desafio tem a ver com a curva de aprendizagem da *framework*. Tempo é dinheiro, e aprender a trabalhar com uma *framework* e tirar partido dela leva tempo, e portanto à partida já se está a incorrer em custos. Nestes casos é necessário ver se os custos de aprendizagem podem ser superados com o uso da *framework*, quer através da qualidade e estabilidade final do produto quer do tempo que se pode poupar ao desenvolver a aplicação. É necessário seguir as convenções da *framework*, podendo estas diferir muito de outras *frameworks* similares, fazendo com que uma mudança de *framework* requeira aprender toda uma nova abordagem. Outro aspeto importante é a integração de várias *frameworks* no mesmo projeto. Uma *framework* raramente tem todas as funcionalidades que se pretendem e neste caso pode-se optar por desenvolver essas funcionalidades ou introduzir outras *frameworks* mais específicas que as cubram. Contudo, em muitos casos revela-se uma tarefa difícil, podem ser uma fonte de problemas e *bugs*, ou em casos extremos duas *frameworks* podem ser de muito difícil integração por seguirem abordagens distintas e incompatíveis [BMM+97].

Em certos casos de aplicações muito específicas que tenham de ter um excelente desempenho ou mesmo responder em tempo real, o uso de *frameworks* é desaconselhado porque não têm um desempenho tão bom como uma aplicação feita de raiz. Isto acontece porque a introdução de uma *framework* vai

adicionar níveis de abstração, que são fundamentais para serem extensíveis e modulares. Para sistemas em grande escala a maioria das *frameworks* não são viáveis, pelo que normalmente nem se consideram usar pois é preciso uma abordagem muito específica e adaptada ao sistema em concreto.[MBF99]

Em relação à segurança, as falhas de segurança ou bugs numa *framework* vão afetar diretamente todas as aplicações que a usam. É portanto um risco que se aceita ao usar uma *framework*. No entanto esta questão é severamente amenizada tendo em conta todo o processo de testes e correção de bugs pelo qual passam as *frameworks* mais conhecidas.

2.4 *Guidelines* para o desenvolvimento de aplicações web móveis

Desenvolver aplicações web para dispositivos móveis requer uma abordagem completamente diferente daquela que se tem ao desenvolver para computadores normais. Existem um conjunto de desafios e particulariedades que se têm que ultrapassar de modo a desenvolver uma aplicação funcional, atrativa e que proporcione uma boa experiência de utilização. Nesta secção vamos analisar príncipios orientadores e boas práticas no desenvolvimento de aplicações web móveis, bem como alguns desafios inerentes a esta área.

2.4.1 Simplificar é fundamental

Uma característica comum à maior parte dos websites adaptados para dispositivos móveis é que estes são versões reduzidas do website dito normal. Esta redução é ao nível de vários factores, tais como a quantidade de informação disponibilizada ou também as funcionalidades suportadas.

2.4.2 Espaços neutros

Com o relativo pequeno tamanho dos ecrãs e com a quantidade de informação que se quer disponibilizar, é fácil acabar com um website muito pouco usável e de difícil interpretação, leitura ou compreensão por parte do utilizador. Isto acontece porque se eliminam espaços neutros ou espaços em branco. Não é por acaso que os websites mais populares e com melhor *design* têm um *layout* simples, e com espaços neutros para que seja agradável ao utilizador.

2.4.3 Evitar imagens

Nestes últimos 4 a 5 anos a velocidade das ligações à internet evoluíram exponencialmente, sendo normal ter-se uma ligação de 20+ Mbps ou, em alguns casos, ter velocidades de 100 ou 200 Mbps. Pode dizer-se que as ligações à internet dos dispositivos móveis também evoluíram, no entanto

são ainda bastante inferiores em vários aspetos. A velocidade de ligação que se consegue nos dispositivos móveis é bastante inferior e tem mais tendência a ser afetada por fatores externos, tais como o número de utilizadores do mesmo *hotspot* ou até a qualidade do sinal. A latência é outro problema que é mais notório nas comunicações móveis. Outro aspecto crítico é o custo dos tarifários de internet móvel, estando estes muitas vezes limitados no consumo de tráfego. Com esta grande largura de banda massificada, as páginas web evoluiram de forma a serem muito ricas em conteúdo visual nomeadamente imagens, icons, banners, etc. Devido às restrições inerentes às comunicações móveis, principalmente em termos de velocidade e limite de tráfego, é de bom senso não utilizar muitas imagens nas páginas web para dispositivos móveis, limitando a sua utilização ao estritamente necessário.

2.4.4 Priorização do conteúdo

Devido à simplicidade que os websites móveis têm que ter, os seus conteúdos são muito priorizados. Quando um utilizador usa pela primeira vez um website móvel fica surpreendido pelo grande grau de priorização a que os conteúdos e funcionalidades foram sujeitos. Todo o conteúdo é racionalizado e é tido em consideração que o utilizador esté num contexto em que a palavra-chave é a mobilidade.

Um aspeto positivo da priorização do conteúdo é a quase ausência de publicidade nas páginas, ao contrário daquilo a que estamos habituados, simplesmente porque não existe espaço. Entre ganhar dinheiro com anúncios publicitários ou ter uma página mais funcional e mais profissional, as empresas optam pela segunda escolha na esperança de ter mais benefícios devido a clientes satisfeitos.

Neste processo de adaptação, houve também a necessidade de se decidir o que é realmente essencial aparecer na versão adaptada de modo às sua páginas não ficarem sobrecarregadas. É um aspeto bastante importante, ainda para mais quando o site normal tem bastante conteúdo diversificado [Sne09].

2.4.5 Contexto de utilização

Existem várias assumpções que se podem fazer quando se está a desenvolver uma aplicação web especificamente para dispositivos móveis. E partindo destes prossupostos podemos aumentar e melhorar significativamente a interactividade entre o utilizador e o dispositivo móvel. Nesta secção falaremos sobre alguns casos onde a usabilidade é melhorada devido ao conhecimento do contexto em que o utilizador se encontra [GT04].

Números de telefone ou contactos

Uma assunção que se pode fazer é que a maioria dos dispositivos móveis têm a capacidade de fazer chamadas telefónicas. Partindo deste princípio, po-

Figura 2.2: *Dois exemplos de diferentes tipos de input*

demos fazer com que todas as ocorrências de números de telefone ou outro tipo de contactos sejam directamente clicáveis e assim possibilitar ao utilizador fazer a chamada para o contacto em que clicou [Raa10]. Por exemplo:

```
1    <a href="tel:78346534" class="phone-link">78346534</a>
```

Tipos de *input*

Atualmente a maioria dos dispositivos móveis não possui um teclado físico, mas sim um teclado virtual que aparece no ecrã sempre que é necessário introduzir algum tipo de informação pelo utilizador, ver figura 2.2. Devido à falta de espaço no ecrã, os teclados são limitados no número de caratéres, símbolos ou números.

A falta de espaço no ecrã leva à existência de um simples teclado "qwerty", o que obriga um utilizador, que pretende introduzir informação, a mudar várias vezes de tipo de teclado para ter acesso a determinados símbolos ou números. Desta forma, para facilitar a vida ao utilizador e para tirar partido das vantagens de se ter um teclado virtual pode-se usar os diferentes tipos de *input* presentes no HTML5, como por exemplo:

```
1    <input type="tel" />
2    <input type="email" />
```

A primeira linha especifica que os dados a introduzir são numéricos, podendo-se observar este tipo de *input* na primeira imagem da figura 2.2. Quanto à segunda linha, é especificado que os dados a introduzir são do tipo "email", o que na prática faz aparecer um teclado "qwerty" acompanhado dos símbolos "@" e o "." (ponto), observando-se isso na segunda imagem da figura 2.2 [Raa10].

Orientação do dispositivo

Uma outra característica que é intrínseca aos dispositivos móveis e que não está presente nos computadores normais (portáteis, desktops) é a capacidade de se mudar a orientação do dispositivo originando dois modos de visualização, o modo *landscape* e o modo *portrait*. Com a possibilidade de a qualquer momento se mudar a orientação do dispositivo móvel e consequentemente o modo de visualização, é bastante útil ter a capacidade de detetar esta mudança para que eventualmente se possam carregar diferentes *stylesheets* consoante o *layout*. Para detetar a orientação do dispositivo, usam-se as seguintes *media queries*:

```
1    @import url("portrait.css") all and (orientation:portrait
         );
2    @import url("landscape.css") all and (orientation:
         landscape);
```

Quando a orientação estiver no modo *portrait*, o *browser* carrega a *stylesheet* correspondente, portrait.css, e o mesmo acontece no modo *landscape*, em que é carregada a *stylesheet* landscape.css. Esta flexibilidade é muito útil porque se pode alterar completamente o *layout* de uma página web para que se ajuste a estes dois modos de visualização [And10].

2.5 Conclusões

O desenvolvimento de aplicações para dispositivos móveis está repleto de desafios, sendo preciso adotar metodologias e abordagens adequadas a este paradigma.

Um dos maiores desafios é certamente o tamanho do ecrã. No desenvolvimento de aplicações para desktops, o tamanho do ecrã não é por si só um desafio, mesmo com as diferentes tamanhos e resoluções que possam existir. No entanto, no desenvolvimento de aplicações para dispositivos móveis, o tamanho do ecrã já se torna um dos maiores desafios devido às suas dimensões reduzidas. Este problema já foi mais acentuado com *smartphones* mais antigos, que tinham ecrãs com menor tamanho e resolução dos atuais. É certo que com o avanço tecnológico, o tamanho e resolução dos ecrãs ainda poderá aumentar mais, embora haja um limite físico, pelo menos no que diz respeito ao tamanho [Bre02].

Outro desafio que existe, é relacionado com a simplificação de conteúdos que é preciso fazer. Decidir o que é essencial é um problema com que muitos *developers* não estão habituados a lidar, porque sempre desenvolveram num ambiente em que se podia ter uma interface muito rica em termos de quantidade de informação. Mas na área móvel, devido ao reduzido tamanho dos ecrãs, todo o espaço é crítico. Sendo assim, cabe ao *designer* da interface escolher criteriosamente qual o conteúdo a disponibilizar, qual a sua disposição visual e que funcionalidades se deve colocar na interface. Tendo como

ponto de partida um website normal, que está repleto de informação e funcionalidades, decidir quais as funcionalidades principais ou que informação é mais importante pode ser uma tarefa bastante difícil, com a agravante de que é necessário ter em conta o contexto em que o utilizador está inserido. Por exemplo, uma companhia ferroviária sabe que sempre que um utilizador acede ao seu site através do seu telemóvel, existe uma grande probabilidade do utilizador querer saber de forma rápida os horários dos comboios. A ideia fundamental a reter é que os cenários de utilização dos dispositivos móveis são muito diferentes daqueles que são encontrados em casa ou nos escritórios das pessoas.

Nesta área em particular, o desenvolvimento tecnológico é rápido e de uma forma constante. Existem muitas variáveis que fazem parte do avanço tecnológico, tendo diferentes impactos no produto final. Temos, por um lado, o avanço ao nível do hardware, com cada vez maior capacidade de processamento, mais espaço de armazenamento, ecrãs com maior resolução e qualidade, etc. Por outro lado, e para tirar partido do hardware, existem os avanços ao nível do software, desde os sistemas operativos que ficam cada vez mais complexos e robustos, até aos browers com suporte às mais recentes tecnologias. É importante que se tenham estas variáveis todas em conta para que se possa tirar total partido dos dispositivos para os quais se está a desenvolver um determinado produto.

Um dos maiores entraves ao desenvolvimento de aplicações para dispositivos móveis é a grande variedade nas suas especificações em termos de hardware, bem como em termos de software (e.g. *browser*, sistema operativo, etc). Devido a toda esta heterogeneidade é praticamente impossível fazer testes consistentes em todas as plataformas. Mesmo que um *developer* conseguisse testar em todas as plataformas existentes no mercado, passado pouco tempo saem novas versões quer de brwosers quer de sistemas operativos. Este problema foi em grande parte colmatado com o aparecimento e desenvolvimento de emuladores. Realizar os testes necessários nos diversos sistemas operativos e nos mais diversos *browsers* foi então possível através dos emuladores. Apesar das grandes vantagens dos emuladores, têm também algumas desvantagens, nomeadamente o facto de em muitos casos não simularem as reais capacidades computacionais ou de memória ou até mesmo nas latências de acesso à internet.

Capítulo 3

Casos de estudo

Como o projeto conducente a esta dissertação consiste em desenvolver uma aplicação web para dispositivos móveis, surgiu a ideia de se estudar e analisar várias aplicações no mesmo âmbito. O objetivo principal era ganhar alguma percepção e espírito crítico no que diz respeito à modelação da interface, para daí retirar ilações que fossem úteis e pudessem ser aplicadas aquando do desenvolvimento da aplicação. Desta forma, neste capítulo vão ser analisadas duas adaptações para *smartphones* de websites mais complexos e portanto não tão otimizados para serem visulizados a partir de pequenos dispositivos móveis. Vão ser analisadas e comparadas as adaptações que foram feitas nas versões para *smartphones* em comparação com as versões normais para desktop, incidindo principalmente no que diz respeito à usabilidade geral e no modo como a informação está organizada nas páginas. Vão ser também analisadas as funcionlidades que tiveram que ser sacrificadas, seja por imposição tecnológica ou devido ao tamanho da página/interface ser muito mais reduzido ou até por não serem relevantes no contexto em que vão ser utilizadas.

Nos dois casos de estudo que se seguem, em primeiro lugar é analisada a versão para desktops do ponto de vista das suas funcionalidades, organização estrutural e quantidade de conteúdo disponível. Depois é analisada a versão para *smartphones* segundo os mesmo critérios de forma a podermos comparar as duas. De referir que os websites que foram alvo de estudo são bastante complexos para serem analisados e descritos aqui na íntegra. Desta forma a análise e descrição vai focar-se principalmente na página inicial (*Homepage*) e nos casos que forem relevantes também irão ser analisadas sub-páginas.

Os casos de estudo selecionados foram o site do *Sapo*[1] e do *Público*[2], com os seus sites adaptados para *smartphones Sapo Mobile*[3] e *Público Mobile*[4], respetivamente.

[1] http://www.sapo.pt
[2] http://www.publico.pt
[3] http://m.sapo.pt
[4] http://m.publico.pt

3.1 Sapo mobile

O primeiro caso de estudo analisado é o *Sapo Mobile*. É uma versão do site do *Sapo* e, como veremos neste capítulo, foi concebido e desenvolvido especialmente para *smartphones*.

Comecemos então por analisar o site do *Sapo*, mais propriamente a sua página inicial, a qual está ilustrada na figura 3.1. À primeira vista podemos ver que é um site muito rico em conteúdo informativo sobre diversos temas da atualidade. A grande quantidade de informação provoca uma utilização total do espaço disponível na página, não comprometendo no entanto a sua legibilidade. Para se analisar melhor a organização e estrutura geral desta página, foi feito um *mockup* (figura 3.2) que realça esses mesmos aspetos. Analisando então o *mockup*, podemos ver que a página começa com um pequeno cabeçalho com vários *links* e uma zona de pesquisas. No resto da página o *layout* é composto por duas colunas, sendo que a coluna da esquerda é a que tem principal destaque e por isso tem também uma maior dimensão do que a coluna da direita. A organização do *layout* em colunas permite condensar o conteúdo de uma forma eficaz e otimizar o espaço. Quanto à coluna da esquerda, é nesta que se concentra toda a informação mais relevante e com teor noticioso sob a forma de pequenos excertos de notícias. Praticamente todo este conteúdo noticioso está concentrado em dois blocos, designados por *tabbed panels*, tal como podemos ver na figura 3.2. O uso de *tabbed panels* é uma forma inteligente de organizar o conteúdo por áreas, como por exemplo os destaques, desporto, economia, tecnologia, cinema, música, etc. Com a organização por áreas e a utilização dos *tabbed panels*, consegue-se disponibilizar muita informação num espaço relativamente pequeno de uma forma dinâmica. O utilizador ao aceder a cada uma das áreas faz com que o conteúdo visível mude dinamicamente de acordo com a área escolhida. Desta forma o utilizador pode aceder de uma forma rápida e eficaz a um grande conjunto de notícias do seu interesse. Depois destes dois *tabbed panels*, temos uma secção com vários *links* para os vários canais do Sapo, ao qual se segue um pequeno mapa com acesso direto aos sites internacionais da Sapo e temos ainda uma pequena pergunta do dia.

A coluna da direita, com um caráter mais secundário, está reservada essencialmente para conteúdo publicitário e para a promoção dos canais de mercado do Sapo, como o canal de emprego, leilões, vendas de carros, etc. Sobre a estrutura e organização desta coluna não há muito a referir, já que esta é relativamente simples. Temos conteúdo publicitário em pequenos blocos reservados para esse efeito, e depois conteúdos relativos aos canais de mercado dispostos verticalmente, com excertos simples de texto a acompanhar pequenas imagens.

Analisemos agora o site *Sapo Mobile*, do qual estão ilustrados na figura 3.3 vários segmentos da sua página inicial. À primeira vista e do ponto de vista de um mero utilizador, pode ver-se que o site está de facto adequado

Figura 3.1: *Página inicial do Sapo*

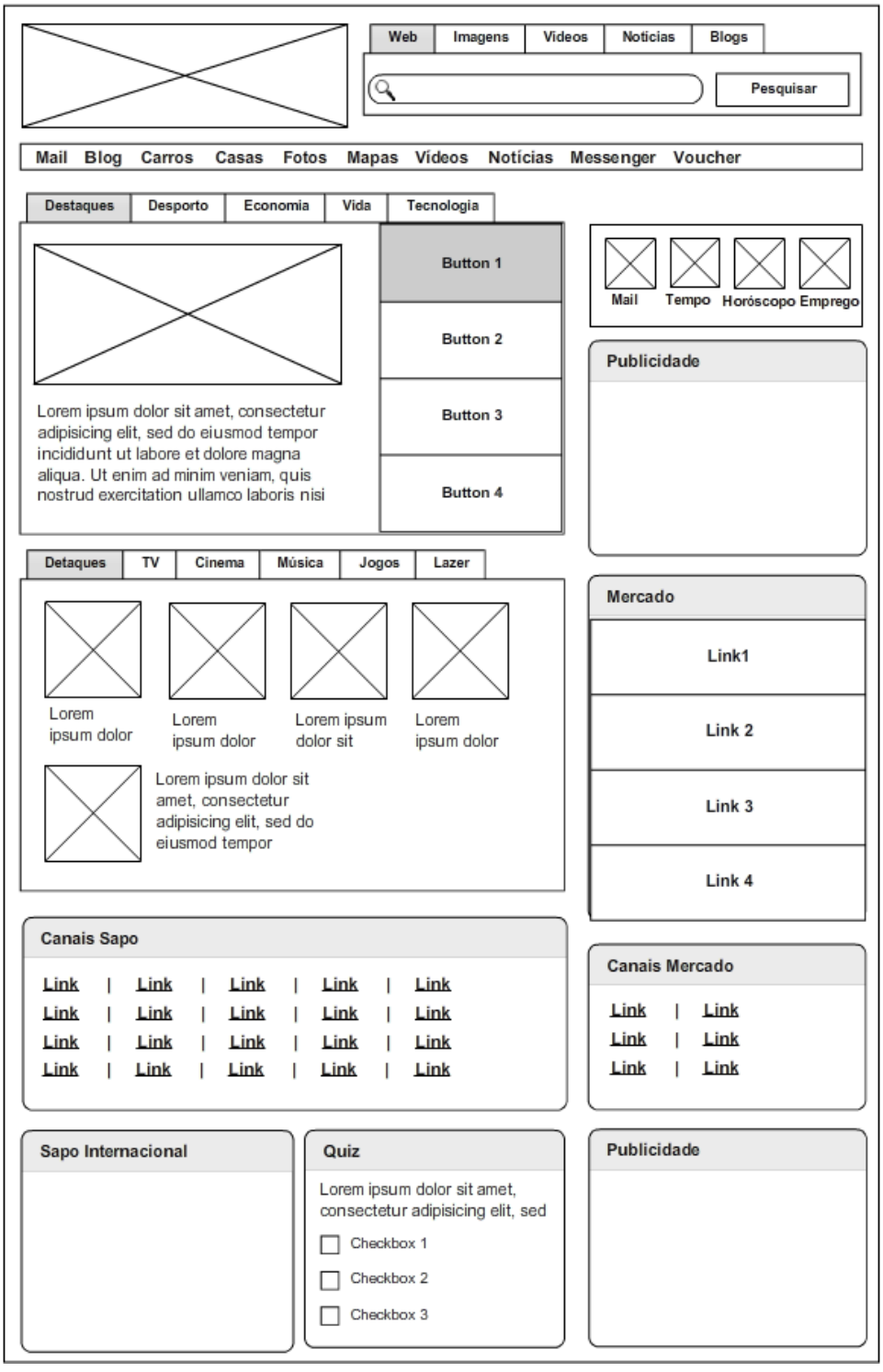


Figura 3.2: *Mockup da página inicial do Sapo*

para ser visualizado através de um *smartphone*. E para isso tiveram que ser feitas mudanças radicais, não só em termos de conteúdo disponível mas também em termos de *layout* da página e outros aspetos importantes no contexto móvel. O *layout* geral da página é constituído por uma única coluna, e o seu conteúdo está organizado em vários blocos ou secções dispostos uns a seguir aos outros. A página começa com uma barra de navegação, onde está também o logotipo do *Sapo Mobile*. O uso desta barra de navegação é bastante comum em aplicações nativas para *smartphones*, sendo que o seu objetivo é proporcionar ao utilizador uma maior facilidade de navegação e um maior sentido de orientação. Logo a seguir à barra de navegação, temos um pequeno *banner* de publicidade. Embora aqui o espaço reservado para a publiciade não comprometa muito a quantidade de informação visível, a página ficaria com um aspeto mais limpo se esta não existisse. Temos depois uma simples caixa de texto destinada a pesquisas. A seguir temos o conteúdo noticioso, disposto numa lista vertical organizada por áreas, na qual cada entrada da lista é composta por uma imagem e um excerto da notícia em destaque da área em questão. Temos logo a seguir a secção da banca de jornais, mais precisamente um *link*, ilustrado com capas de jornais, para a página onde estão as manchetes dos vários jornais nacionais. De salientar que no site normal, a banca de jornais quase passa despercebida, enquanto que na versão mobile já tem um bom destaque. De certa forma é um conteúdo que se enquadra bem no contexto onde vai ser visualizado o site. É uma clara priorização de conteúdo, em que o contexto de utilização do site tem um peso determinante no conteúdo disponível. A seguir, uma pequena secção destinada ao futebol, apenas com dois *links* para os resultados e classificações. Depois temos a secção multimédia, na qual foi adotada uma abordagem muito semelhante à secção inicial das notícias. A seguir uma pequena secção do sapo *voucher*, que publicita um determinado produto e que serve também de *link* para a própria página do sapo *voucher*. Na parte final da página temos um menu formado por várias imagens dispostas numa matriz. Este menu serve para de uma forma rápida o utilizador tenha acesso a várias secções que não estão representadas na página inicial, mas que são também importantes. Desta forma, a página inicial não é muito complexa em termos de informação, no entanto dá acesso direto às várias secções do site. Em relação a esta versão adaptada para *smartphones*, está então concluída a sua análise geral.

Vamos agora comparar e analisar as adaptações feitas na versão para *smartphones* a partir da versão normal do site. Em primeiro lugar, o *layout* da página foi alterado e adaptado ao tamanho típico de um ecrã de um *smartphone*. Passamos de um *layout* de duas colunas, onde era previlegiada a quantidade de informação visível, para um *layout* de uma coluna simples. Em segundo lugar, a quantidade de informação disponível ao utilizador logo na primeira página reduziu drasticamente. Para além da redução da quantidade de informação, houve também alguma priorização do conteúdo uma vez

Figura 3.3: *Sapo mobile Homepage*

que o contexto de utilização do site normal e para *smartphones* é bastante diferente.

Na versão normal temos uma pesquisa mais dinâmica, sendo que se pode escolher o tipo de conteúdo que se está a pesquisar como por exemplo imagens, vídeos, notícias, blogs, etc. Por sua vez, a pesquisa na versão mobile foi extremamente simplificada ao ponto de só termos uma caixa de texto e o botão para submeter a pesquisa. De certa forma é uma simplificação necessária e adequada, primeiro porque as pesquisas não devem ser a principal funcionalidade utilizada e em segundo lugar porque há pouco espaço para uma secção de pesquisas por tópicos ou áreas.

Em relação ao conteúdo noticioso, as adaptações feitas foram principalmente no sentido de simplificar e reorganizar o conteúdo. O *tabbed panel* usado na versão normal dá lugar a uma lista vertical de *links* com pequenas imagens e excertos de notícias de cada área. Na versão normal, cada painel correspondia a uma área (e.g. desporto, economia, tecnologia, etc) e continha vários excertos de notícias, sendo que na adaptação é apenas previsualizado uma notícia de cada área. Apenas se seguirmos o *link* de uma área é que temos acesso ao resto das notícias dessa área. Desta forma houve uma adaptação da forma como o conteúdo é visualizado, e também houve uma simplificação do conteúdo apresentado em cada área das notícias. Através destes dois passos, conseguiu-se adaptar de uma forma eficaz um conteúdo complexo de modo a poder ser visualizado num *smartphone*.

Quanto à publicidade, esta está presente nas duas versões do site, mas em proporções diferentes. Na versão normal, a publicidade está presente em dois blocos quadrados que ocupam a largura total da coluna da direita, sendo por isso bem visíveis. Na versão adaptada para *smartphones*, a publicidade aparece sob a forma de um simples banner no topo da página. Embora nesta versão o ideal fosse não existir espaço destinado à publicidade, o facto é que

esta é feita de uma forma subtil e não ocupa muito espaço, pelo que não compromete uma boa experiência de utilização.

No que diz respeito à priorização do conteúdo, temos o caso da banca de jornais. No *site* normal, temos acesso à banca de jornais através de um pequeno *link* quase sem destaque nenhum. No *site* adaptado a banca de jornais ganhou um espaço com bastante destaque. De certa forma é um conteúdo que se adequa ao contexto móvel.

Neste processo de adaptação, algumas secções foram cortadas com a finalidade de não sobrecarregar a página inicial. Das que foram eliminadas da página inicial, algumas secções foram delegadas para outras páginas enquanto que outras simplesmente não estão presentes na versão adaptada. As secções eliminadas são o sapo mercado, sapo internacional, canais sapo e a pergunta do dia. Estas secções foram eliminadas porque, provavelmente, não se enquadram bem no contexto móvel.

3.2 Público Mobile

O segundo caso de estudo a ser analisado é o *Público mobile*. De forma análoga ao Sapo Mobile, é uma versão do *site* do *Público* desenvolvida e adaptada para ser visualizada em *smartphones*. O *Público* é um jonal diário nacional bastante conhecido, e o seu *site* segue a mesma vertente jornalística.

Comecemos então por analisar o *site* do *Público*, mais propriamente a sua página inicial, a qual está ilustrada na figura 3.4. Importa realçar que na figura 3.4 não está representada a página inicial inteira, visto esta ser bastante comprida. De qualquer forma, a parte que foi cortada era a continuação das 3 colunas e continha mais notícias, imagens, publicidade e portanto mantinha a estrutura e organização da parte visível na figura. À primeira vista podemos ver que é um *site* com bastante informação, ilustrado com várias imagens referentes às notícias com maior destaque e de certa forma um pouco confuso para um utilizador que o visite pela primeira vez. Para além do *site* fazer uso pleno do espaço horizontal com notícias, é também bastante longo disponibilizando uma quantidade massiva de excertos de notícias e outros conteúdos. Mais uma vez, para se analisar melhor a organização e estrutura geral desta página foi feito um *mockup* (figura 3.5), que realça esses mesmos aspetos. Analisando então o *mockup*, podemos ver que a página começa com um pequeno cabeçalho com o logotipo do *Público* juntamente com umas pequenas imagens e respetivos excertos de notícias. Ainda no cabeçalho, temos uma lista horizontal com vários *links* para áreas de interesse específicas como, por exemplo, a política, economia, desporto, sociedade, etc. Depois, temos um *layout* composto por três colunas com dimensões semelhantes. Das três colunas, as duas primeiras contêm maioritariamente excertos de notícias ou artigos, enquanto que a terceira coluna está reservada principalmente para publicidade juntamente com um artigo de opinião e o *plugin* do

Facebook. Nas duas primeiras colunas, o conteúdo está disposto na vertical e aparentemente sem uma organização por áreas. Quanto à terceira coluna, a publicidade aparece em imagens intercaladas com outro conteúdo como, por exemplo, o artigo de opinião, o *plugin* do facebook ou informação das notícias mais lidas, mais comentadas ou partilhadas. A organização do *layout* em colunas permite de uma forma eficaz condensar muito conteúdo e otimizar o espaço. Na parte final da página, temos uma secção com vários *links* para áreas de interesse específico. Estes *links* estão dispostos em matriz para uma fácil percepção e são de certa forma redundantes pois já estão também presentes no cabeçalho, embora aqui lhes seja dado mais relevo e destaque. Depois, temos a secção *Público Digital* que publicita as várias vertentes tecnológicas onde é possível consultar o *site* do *Público*, como por exemplo em *tablets*, no kindle ou em *smartphones*. Em relação à versão da página normal, ou para *desktops*, está concluída a sua análise geral.

Analisemos agora o site *Público Mobile*, do qual estão ilustrados na figura 3.6 vários segmentos da página inicial. À primeira vista e do ponto de vista de um mero utilizador, pode-se ver que o *site* está de facto adaptado para ser visualizado através de um *smartphone*. Houve mudanças radicais tanto em conteúdo disponível como na organização e estrutura geral do *layout* da página. Quanto ao *layout* geral da página, este é constituído por uma única coluna. O conteúdo está organizado em vários blocos com excertos de notícias, dispostos verticalmente uns a seguir aos outros. A página começa com uma barra inicial, onde está o símbolo do *Público Mobile* e mais três símbolos à direita. Esses três símbolos servem, respetivamente da esquerda para a direita, para aceder diretamente à página inicial, para fazer com que a página não carregue imagens e para aceder à edição impressa do jornal. Esta barra inicial mantém-se inalterada independentemente da página em que estamos, não fazendo portanto a função de barra de navegação do *site*. Logo a seguir, temos outra barra vermelha que tem a funcionalidade de um menu. Ao clicar-mos sobre esta barra, o menu expande-se e temos acesso a uma lista com várias áreas de interesse como por exemplo política, economia, cultura, sociedade, desporto, etc. De forma análoga à barra inicial, esta também se mantém inalterada independentemente da página que se está a ver, proporcionando assim um acesso rápido a outras áreas de interesse. A seguir vem uma secção de destaques, com vários excertos de notícias acompanhadas de pequenas imagens. Esta secção começa com uma imagem a ocupar toda a largura da página e com um pequeno texto sobreposto referente à notícia que mais destaque se quer dar de entre os destaques. Temos depois outros excertos das notícias em destaque numa lista vertical, em que cada entrada é um *link* para a página com a notícia completa. Depois desta secção de destaques, seguem-se as seguintes secções: últimas notícias, notícias mais lidas e últimos videos. Estas secções em termos de estrutura e organização do conteúdo são muito idênticas à secção de destaques, embora tenham a particularidade de estarem inicialmente minimizadas, como se pode obser-

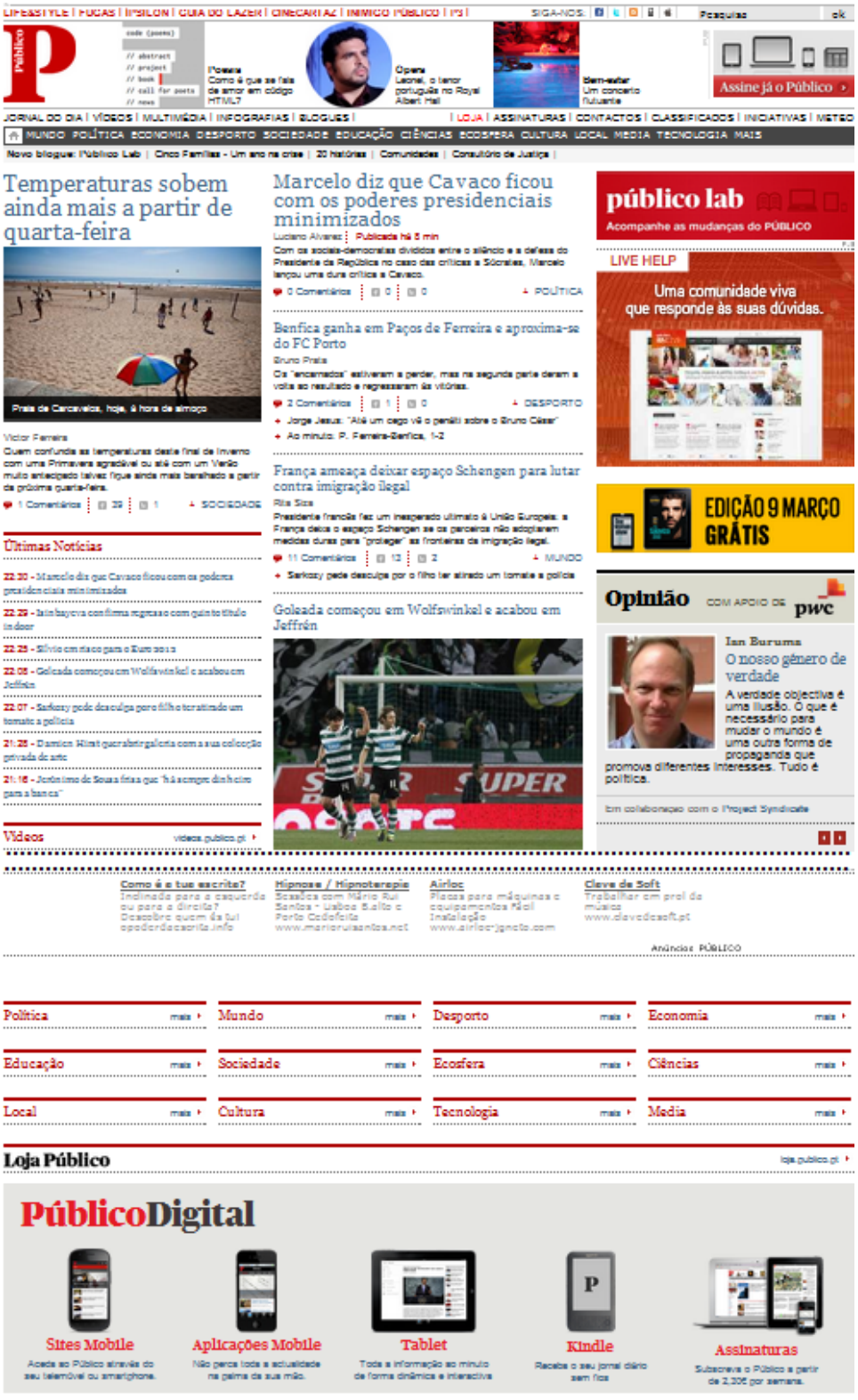


Figura 3.4: *Página inicial do Público*

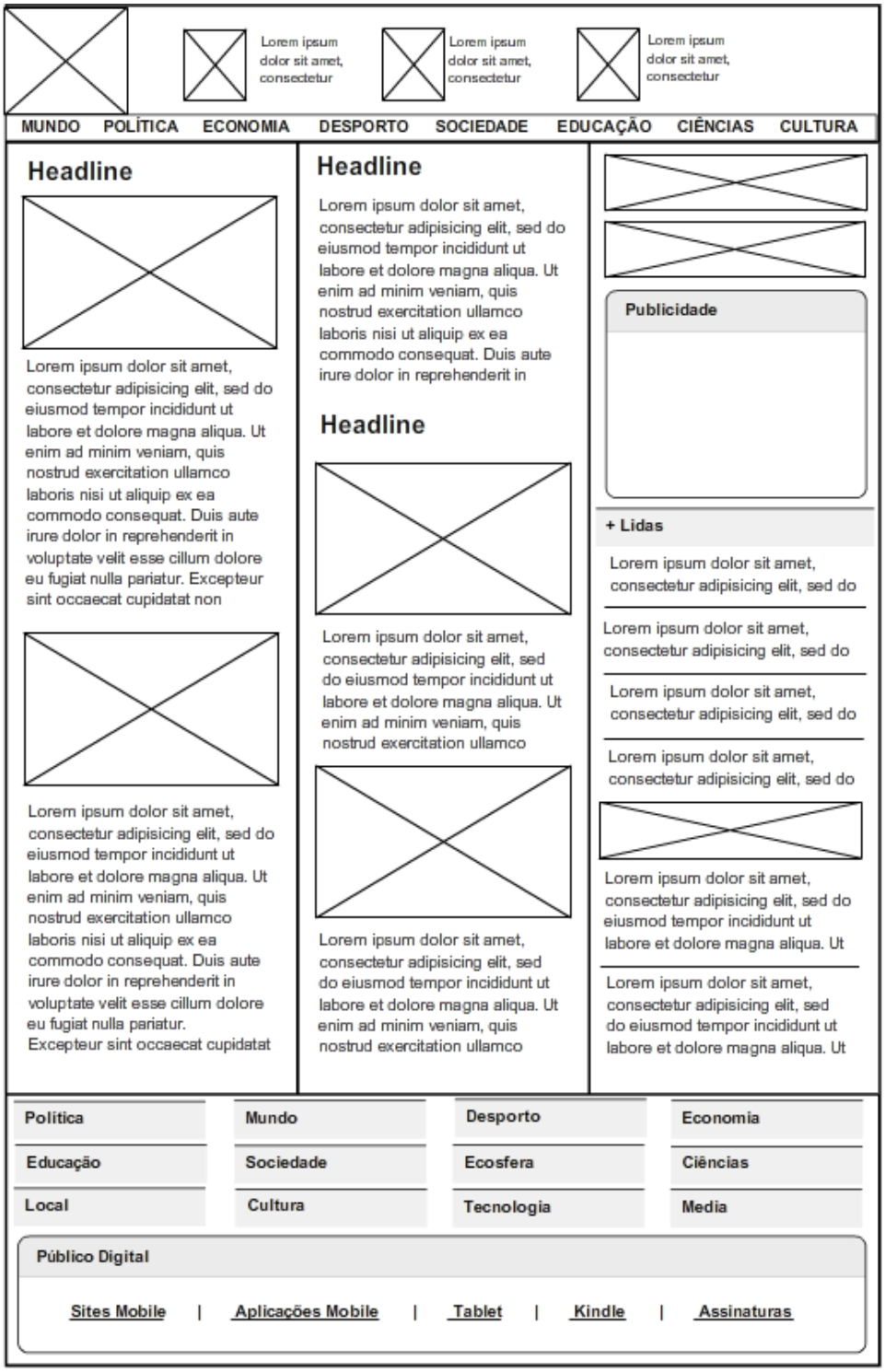


Figura 3.5: *Mockup da página inicial do Público*

Figura 3.6: *Público mobile Homepage*

var na terceira imagem da figura 3.6. Já na parte final da página, temos uma pequena lista vertical com três *links* que redirecionam para páginas que não têm própriamente a ver com notícias, mas sim relacionadas com lazer. Temos ainda uma zona de pesquisas, com uma simples caixa de texto e o botão para efetuar a pesquisa.

Vamos agora comparar e analisar as adaptações que foram feitas na versão para *smartphones* a partir da versão normal do *site*. A mudança mais óbvia que foi feita diz respeito ao *layout*, em que há uma alteração radical para que se adeque ao tamanho típico de um ecrã de um *smartphone*. Passamos de um *layout* de três colunas, onde era priveligiada a quantidade de informação visível, para um *layout* de uma coluna simples. Outra mudança profunda que ocorreu, foi a diminuição drástica da quantidade de informação logo disponível ao utilizador. Houve também uma certa priorização do conteúdo ou notícias selecionadas, uma vez que o contexto de utilização nas duas versões é bastante diferente.

Quanto à publicidade, esta está presente nas duas versões, mas em proporções diferentes. Na versão normal, a publicidade está presente essencialmente na terceira coluna sob a forma de vários *banners* retangulares e quadrados, que ocupam a largura dessa coluna, sendo por isso bem visíveis. Na versão adaptada para *smartphones*, a publicidade aparece sob a forma de um simples *banner* retangular que aparece entre as notícias de destaque.

3.3 Conclusões

Da análise efetuada podem retirar-se várias conclusões acerca da forma como as páginas web foram adaptadas para serem visualizadas em ecrãs de pequenas dimensões, que geralmente se encontram em *smartphones*.

De uma perspetiva geral, ambas as adaptações têm algumas características em comum, como por exemplo: o *layout* da página, a quantidade de conteúdo e imagens visíveis ou até a barra de navegação no topo da página. Um facto curioso é que ambos os sites nas suas versões para desktop, seguem políticas diferentes no modo como têm a sua página organizada, mas depois ao fazerem as respetivas adaptações, acabaram com um *layout* semelhante e com bastantes aspetos em comum. Esta discrepância entre as versões para desktop e a semelhança entre as versões para *smartphones* não é exclusivo dos *sites* analisados. De certa forma isto acontece devido ao grau de liberdade ser muito menor quando se modela e implementa um site para *smartphones*, acabando por usar-se os mesmos príncipios de usabilidade. Ao contrário, nas versões para desktop o grau de liberdade é maior, o que acaba por permitir uma heterogeneidade maior no que diz respeito às suas interfaces.

No capítulo anterior foram mencionadas algumas *guidelines* (boas práticas) para o desenvolvimento de aplicações web para dispositivos móveis. Da análise de vários sites e em particular dos dois aqui abordados, pode-se concluir que há de facto uma certa aproximação entre a teoria e a prática, no sentido do cumprimento com a generalidade dessas *guidelines*. Dessas *guidelines*, as que mais são postas em prática é a simplificação e priorização do conteúdo.

Como foi referido no início do capítulo, um dos objectivos desta análise era adquirir alguma percepção e espírito crítico relativamente à modelação da interface de aplicações web para *smartphones*, com vista à aplicação desses conhecimentos no projeto. Portanto, é importante referir que estas análises foram de facto úteis e importantes aquando da execução prática do projeto.

Capítulo 4

Mobile Monitor: Análise e concepção

Este e os dois capítulos seguintes são dedicados exclusivamente ao projeto que serviu de base a esta dissertação de mestrado e que motivou todo o estudo na área do desenvolvimento de aplicações web para *smartphones*. Tal como se pode aferir pelo título deste capítulo, o nome dado à aplicação desenvolvida foi "Mobile Monitor", uma vez que é uma aplicação de monitorização para dispositivos móveis.

Neste capítulo vamos analisar os requisitos detalhados que foram propostos. Segue-se a concepção e modelação da interface tendo em conta os requisitos propostos. Em relação à concepção da interface, irão ser analisados os *mockups* realizados e o diagrama de estados, que específica todos os estados da interface bem como as transições possíveis entre as várias páginas (estados) que a compõem. Depois é analisada a arquitectura da aplicação, onde é explicado o seu funcionamento geral e como é que as várias camadas aplicacionais interagem entre si.

4.1 Requisitos detalhados

Nesta secção vamos analisar os requisitos mais aprofundadamente, tendo como base os requisitos genéricos apresentados no capítulo 1, secção 1.2. Sendo assim, a aplicação a desenvolver deve ter 5 páginas sendo elas: página de login, página inicial/homepage, página com a listagem de players, página com detalhes de um player e por último a página com os logs de um player. De seguida são especificados mais detalhadamente os requisitos de cada uma destas páginas, nomeadamente o conteúdo e funcionalidades que cada uma deve conter.

Página de login
A página de login deve conter o logotipo da Ubisign, o nome da aplicação,

e o formulário de login (campos *username* e *password*). O formulário deve impedir que se submetam dados inválidos ao servidor. Por dados inválidos entenda-se os campos username ou password vazios. Caso o servidor retorne falha de autenticação, o formulário deve exibir feedback informativo da falha.

Página inicial/homepage

A página inicial deve dar imediatamente uma vista generalizada sobre o estado das redes de *players*, com realce para eventuais casos de falha. A melhor forma de o conseguir é através de uma representação de um gráfico em *pie chart* dos players registados:

- Players a reportar running status "Playing" (verde)
- Players a reportar running status "Stopped" (amarelo)
- Players a reportar running status "Not running" (vermelho)
- Players a reportar running status "Unknown" (cinzento)

O gráfico deve ter associado texto que indique a quantidade exata de Players em cada um dos estados, assim como a quantidade total de Players registados. Cada um dos estados representados deve ter associado um *link* que faça navegar para a página da listagem de Players com o respetivo filtro.

Página com a listagem de players

Esta página deve exibir a listagem dos Players registados no domínio, ordenados alfabeticamente. Devem ser disponibilizados filtros que permitam reduzir o tamanho da listagem em função de:

- Nome da rede (network)
- Running status
- Channel status
- Nome do canal sintonizado

Cada elemento da listagem deve conter a seguinte informação:

- Nome do Player
- Running status
- Nome do canal sintonizado
- Channel status

A listagem deve ser paginada, sendo preciso identificar qual o número de itens default razoável e deve poder ser ordenada pelo nome do player, running status, ou nome do canal sintonizado.

Página com detalhes de um player

Esta página deve permitir ver toda a informação relevante sobre um player, sendo essa informação constituída pelos seguintes campos:

- nome do Player
- nome da rede (network) a que pertence
- estado do player (running status)
- last check-in
- nome do canal sintonizado
- estado do canal (channel status)
- timestamp do início da reprodução do channel
- link para a visualização dos logs de execução

Para além da visualização destas informações deve ainda ser possível ao utilizador alterar o canal sintonizado pelo player.

Página com os logs de um player

Esta página deve exibir uma lista com os logs de execução de um player. Cada entrada na lista dos logs deve conter os seguintes campos:

- Date/Time
- Event
- Description

Os logs de execução do Player irão resumir-se aos eventos do Player (e.g. startup, shutdown, etc.). Portanto, só haverá necessidade dos seguintes filtros:

- intervalo de datas
- intervalo horário

A listagem dos logs deve ser paginada, sendo preciso identificar qual o número de itens default razoável e deve poder ser ordenada pela data/hora.

Dispositivos móveis

A aplicação deverá executar correctamente (admitindo ligeiras diferenças de estilo), nos *browsers* que vêm por omissão nos seguintes dispositivos móveis:

- iPhone, com o Safari
- telemóveis Android-based
- telemóveis Windows Phone-based, com o Internet Explorer 9

A concepção teve como ponto de partida os requisitos propostos e o primeiro passo foi fazer o diagrama geral de Use Cases, que pode ser visto na figura 4.1. Este diagrama contempla, de uma forma simples e intuitiva, as principais funcionalidades que a aplicação tem que ter. Depois, foi feita a especificação textual de cada um desses Use Cases, que descrevem de uma forma simplificada as interações entre o utilizador e o sistema. Estes Use Cases textuais podem ser encontrados em anexo nas figuras 2, 5, 3, 1 e 4.

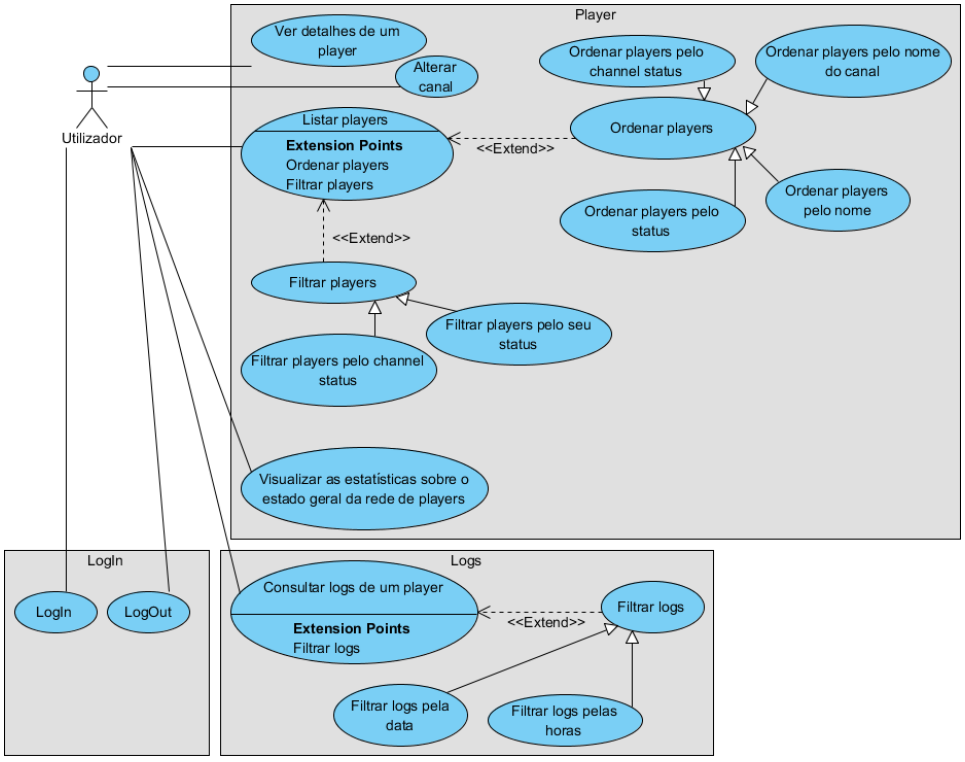


Figura 4.1: *Diagrama geral de Use Cases*

4.2 Concepção e modelação da interface

A interface da aplicação desenvolvida é uma interface web, ou seja, a aplicação vai correr num *browser*, tal como foi solicitado nos requisitos da aplicação. Apesar de ser uma interface web, o objectivo é que a aplicação se pareça com uma aplicação nativa em relação à sua usabilidade e aparência.

Para este propósito recorreu-se à *framework* jQTouch, que já foi descrita no capítulo do estado da arte. A escolha da *framework* recaiu sobre o jQTouch porque esta é bastante versátil, de fácil utilização e com bastante documentação *online*. Para além destas características que abonaram a seu favor, a framework jQTouch tinha a cappacidade de simular a interface de uma aplicação nativa, o que era um requisito para a escolha da *framework*.

A modelação e especificação da interface foi inicialmente feita com o recurso a *mockups*, que são apenas esboços gráficos do que se pretende que seja a interface. Desta forma, os *mockups* serviram para, de uma forma rápida e eficaz, prototipar a interface e validá-la junto do cliente. A ferramenta utilizada para a realização dos *mockups* foi o *mockflow*[1], que se revelou adequada para o que era pretendido.

Os *mockups* da modelação da interface foram divididos em duas etapas, em primeiro lugar fizeram-se os *mockups* de baixo detalhe (baixa fidelidade) e depois os *mockups* de alto detalhe (alta fidelidade). A primeira etapa consistia em fazer *mockups* pouco detalhados e com realce para os requisitos mais importantes e prioritários. Esta etapa tinha como objetivo principal poder rapidamente validar com o cliente os traços gerais da interface da aplicação e obter *feedback*. Na figura 4.2 podemos observar os *mockups* relativos à especificação do login, que contemplam os cenários de erro com as respectivas mensagens.

Na figura 4.3 podemos observar o resto dos *mockups* de baixo detalhe. Aqui, a interface foi prototipada partindo do design pattern *Accordion* [Wil12], que é bastante comum em aplicações web, em que o objetivo era ter acesso rápido a 4 secções principais. Estas secções estão designadas como "Home", "Networks", "Graphs and statistics" e "Geolocalization". De acordo com esta proposta inicial, a secção "Home" teria como conteúdo um pequeno relatório da atividade recente da rede geral de *players*, bem como os alertas dessa mesma rede. A secção "Networks" teria uma listagem das diferentes redes de modo a termos um acesso rápido aos *players* de cada uma dessas redes. A secção "Graphs and statistics" teria um gráfico e informação relativa ao estado geral das redes de *players*. Finalmente, a quarta secção "Geolocalization" teria um mapa a localização dos vários *players* e também informação acerca do seu estado, o que seria útil para eventuais equipas técnicas a operar no terreno a fazer manutenção. Esta modelação inicial, passou por um processo de validação pelo cliente e sofreu algumas mudanças que foram tidas em consideração na especificação e prototipagem dos *mockups* de alto detalhe. Algumas dessas mudanças tiveram que ocorrer devido à aplicação não poder ter acesso a certas informações, nomeadamente à actividade recente e aos alertas. Quanto à geolocalização dos *players*, de modo a fazer o seu mapeamento, também não seria possível fazer isso porque os *players* não tinham associados a si a informação necessária (coordenadas geográficas).

[1] http://www.mockflow.com/

Figura 4.2: *Mockups de baixo detalhe: Login*

Na segunda etapa, a realização dos *mockups* de alto detalhe já teve em consideração a validação e as sugestões dadas pelo cliente na etapa anterior. Desta forma, a interface mudou consideravelmente como se pode observar nas figuras 4.4 e 4.5. Passou ser a uma interface mais simples e com menos funcionalidades e conteúdo informativo (atividade recente e alertas da rede de *players* não foram incluídos), mas mais organizada e desta forma mais semelhante a uma aplicação nativa. A página inicial permite ao utilizador visualizar o estado geral da rede de *players* através de um gráfico "pie chart" e tem também informação relativo ao número de *players* em cada estado ("Playing", "Stopped", "Not Running" e "Unknown") e a sua percentagem em relação à rede geral. Desta página podemos navegar para a listagem dos *players* através de 5 links distintos, que irão listar os *players* com diferentes

Figura 4.3: *Mockups de baixo detalhe: Páginas principais*

critérios. Podemos seguir o link que indica o número total de *players* na rede e vão ser listados todos os *players* sem restrições. Por outro lado, podemos seguir os links onde são indicados quantos *players* estão em cada estado e vão ser listados os *players* que estejam naquele estado. A partir da listagem dos *players*, o utilizador pode visualizar os detalhes de cada um clicando sobre ele. Na página dos detalhes de um player devem estar contidas as seguintes informações: "Network", "Channel", "Running status", "Channel status" e "Last check-in", para além de um link para poder visualizar os logs associados a esse player. Nesta página deve ser ainda possível alterar o canal sintonizado

pelo *player*, pelo que a informação sobre o canal sintonizado deverá aparecer num *drop-down menu* HTML, podendo desta forma este ser alterado. Esta página pode ser observada na primeira imagem da figura 4.5.

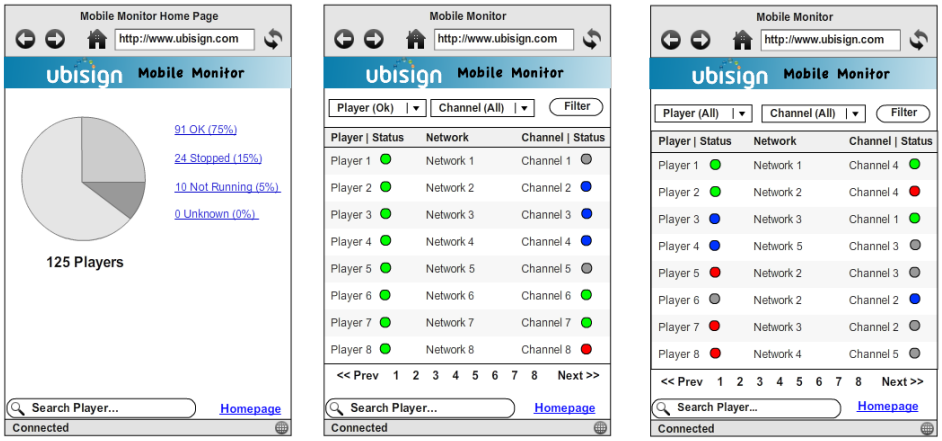


Figura 4.4: *Mockups de alto detalhe*

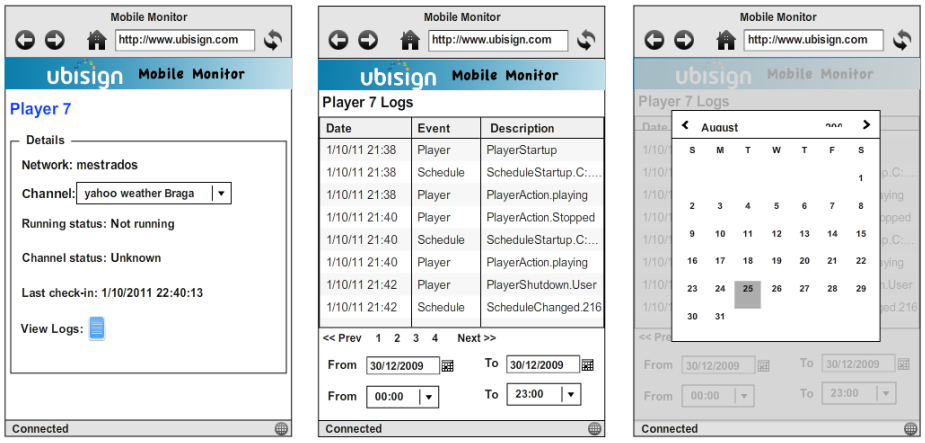


Figura 4.5: *Mockups de alto detalhe (Continuação)*

A página da visualização dos logs pode ser observada na segunda imagem da figura 4.5. Aqui os logs estão dispostos numa tabela com três colunas relativas aos campos relevantes de cada log. Devido à possibilidade de haver um grande número de logs, estes têm que ser paginados. Pela mesma razão, também há a possibilidade de aplicar filtros aos logs. Os filtros possíveis de aplicar são relativos ao intervalo temporal e ao intervalo horário em que os logs são registados. O intervalo temporal serve para quando um utilizador pretende visualizar logs que aconteceram entre duas datas. O intervalo horário serve para definir uma região temporal do dia, por exemplo das 9h às

19h, em que só os logs registados nessas horas é que são visualizados.

Ainda na especificação da interface, foi feito um diagrama de estados (figura 4.6) onde são especificados os possíveis estados da interface, bem como as transições possíveis entre as várias páginas.

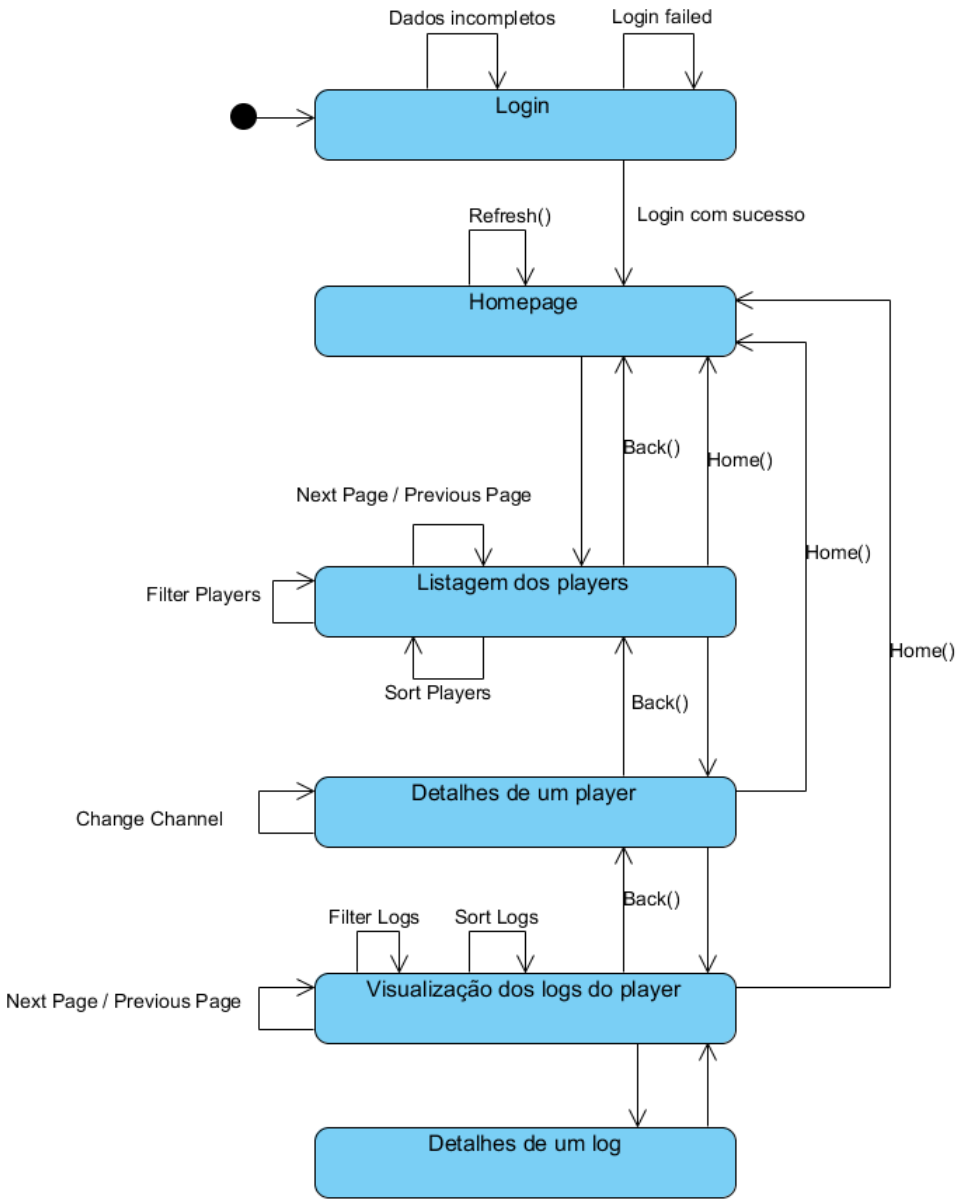


Figura 4.6: *Diagrama de estados da aplicação*

Pelo diagrama podemos ver que existem 6 estados, que correspondem às 6 páginas da aplicação. O utilizador inicialmente é direcionado para a página do *Login* ou para a *Homepage* consoante já estiver autenticado no sistema

ou não. A partir da *Homepage* o utilizador pode clicar no botão de *refresh* para a aplicação voltar a calcular as estatísticas visualizadas nessa página. Neste caso a aplicação irá ficar no mesmo estado/página, sendo que apenas o conteúdo se pode eventualmente modificar. Depois, a partir da *Homepage* podemos transitar para a listagem dos *players*. Nesta página podemos aplicar um novo filtro à listagem dos *players*, podemos alterar a ordem pela qual os *players* estão ordenados na lista, podemos ainda navegar pela paginação da lista através de dois mecanismos, sendo eles os botões "Next page"/"Previous page" ou através de um movimento gestual no ecrã designado por "swipe right"/"swipe left". A partir da listagem dos *players* podemos selecionar um deles e passamos para a página dos detalhes do player selecionado. Esta é uma página informativa e a única função presente é a mudança do canal sintonizado pelo player. A partir desta página podemos seguir um link para visualizar os logs desse player, passando assim para outra página. Na página da visualização dos logs do player temos funcionalidades semelhantes à da listagem dos *players*, embora com algumas diferenças. Essas diferenças são em relação ao tipo de filtros que se podem aplicar bem como a ordenação que é pela data da ocorrência de cada registo. A paginação, tal como na listagem dos *players*, pode ser percorrida através dos botões "Next page"/"Previous page" ou através do movimento gestual no ecrã designado por "swipe right"/"swipe left". Temos ainda outro estado que é referente aos detalhes de um registo/log da listagem dos logs. Tal como está referenciado no diagrama, sempre que se carrega num registo da lista dos logs, é adicionada uma camada com os detalhes desse registo. Podemos verificar que estando em qualquer estado, exceto nos detalhes de um log, podemos ir diretamente para a página inicial ou para a página anterior, o que favorece a navegabilidade na aplicação.

4.3 Arquitectura da aplicação

Nesta secção vamos abordar a arquitetura da aplicação desenvolvida. Na figura 4.7 está representado um esquema relativo à arquitetura da aplicação. Podemos ver que é uma aplicação típica baseada no modelo das 3 camadas, sendo que a camada do acesso aos dados tem algumas particulariedades que serão discutidas. A interface é controlada e criada utilizando a framework jQtouch em conjunto com outras classes JavaScript, sendo estas responsáveis por controlar o conteúdo das páginas. Na camada da lógica de negócio estão as classes e entidades responsáveis por todas as funcionalidades presentes na aplicação.

A camada de acesso aos dados tem uma classe com os métodos disponíveis para ler e escrever na base de dados. Estas leituras e escritas não são feitas diretamente na base de dados, mas sim solicitadas ao serviço Ubisign através de web services. Quanto à camada de dados foi esta a possível arquitetura

uma vez que o acesso direto à base de dados não era permitido.

A utilização de bibliotecas JavaScript, tal como a YUI e jQuery, apenas serviram para facilitar algumas tarefas, como por exemplo a manipulação de dados JSON provenientes do serviço Ubisign.

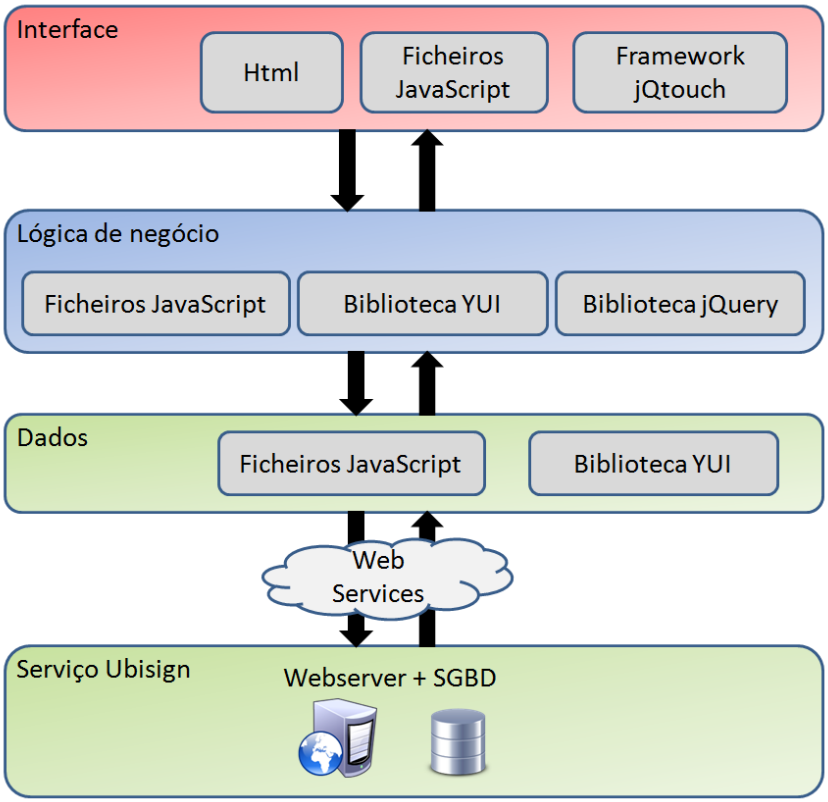


Figura 4.7: *Arquitetura da aplicação*

4.4 Conclusões

Neste capítulo foi, numa primeira fase, efetuado o levantamento de requisitos para a monitorização e controlo de uma rede de *digital signage*. Após esse levantamento, foi apresentada a concepção da aplicação Mobile Monitor, que responde aos requisitos identificados. A aplicação descrita é um exemplo concreto de uma aplicação web com capacidade para fornecer aos utilizadores uma experiência de utilização semelhante às aplicações nativas. Isso é conseguido através do uso da framework jQTouch que proporciona tal experiência de utilização. No próximo capítulo é descrita a implementação da aplicação mobile monitor.

Capítulo 5

Implementação

Neste capítulo vamos analizar a implementação da aplicação ao nível da lógica de negócio e ao nível da camada de dados.

Em relação à especificação da lógica de negócio, a sua modelação foi feita através de diagramas de sequência e de classes. Nestes diagramas são especificadas as várias funcionalidades da aplicação, e também as suas classes juntamente com os seus métodos e atributos. Quanto à especificação da camada de dados, irá ser descrito o mecanismo de acesso à base de dados e as operações permitidas.

5.1 Lógica de negócio

A modelação e concepção da lógica de negócio da aplicação foi feita com base em diagramas UML, sendo eles o diagrama de classes e o diagrama de sequência.

Começou-se por fazer o diagrama de classes geral que serviu para descrever as entidades que iriam compor o sistema e quais as suas associações. Depois, o diagrama de classes foi complementado com todos os métodos e atributos de cada classe. Nos diagramas de sequência, o objetivo é descrever formalmente as funcionalidades da aplicação, onde se especificam as interações e trocas de mensagens entre as várias entidades e camadas do sistema.

O diagrama de classes pode ser visualizado na figura 5.1. Em relação à abordagem seguida, optou-se pelo padrão de desenvolvimento MVC (Model-View-Controller). É uma abordagem que facilita o desenvolvimento e estruturação da aplicação, uma vez que há uma separação clara entre a interface e a camada lógica. Temos portanto umas classes que são os controladores (*controllers*), outras classes que são as vistas (*views*) e temos ainda os modelos (*models*), para além de outras classes auxiliares. Os controladores são o "cérebro" da aplicação. Estes tratam dos pedidos ou qualquer evento que seja lançado pela aplicação, e ficam encarregues de todo o procedimento lógico. As vistas, por sua vez, são as classes que fazem parte da camada da

interface e portanto são responsáveis pela geração das páginas da aplicação. Quanto aos modelos, estes encapsulam informações sobre os vários objetos do sistema e são utilizados tanto pelos controladores como pelas vistas como meio de troca de informação.

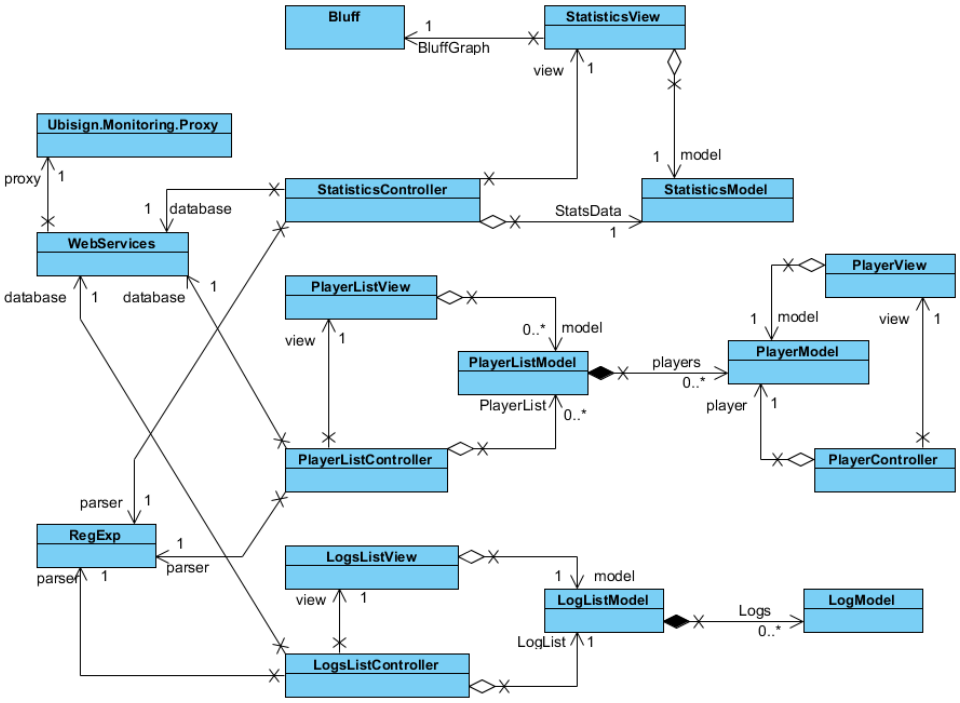


Figura 5.1: *Diagrama de classes*

Partindo do diagrama de classes da figura 5.1, para cada classe foram especificados os seus atributos e métodos, sendo que as associações entre classes já aí estavam presentes. Uma vez que o diagrama de classes completo é demasiado grande para ser visualizado numa única figura, vamos analisá-lo por partes.

Comecemos então por analisar as classes *StatisticsController*, *StatisticsView*, *StatisticsModel* e *Bluff*, que são responsáveis pela página inicial onde se visualizam as estatísticas gerais da rede de *players*. Estas classes, bem como os seus métodos, atributos e associações, podem ser visualizados na figura 5.2. No diagrama salta logo à vista o modelo MVC, onde o controlador é a classe *StatisticsController*, a vista é a classe *StatisticsView* e o modelo é a classe *StatisticsModel*. No controlador, temos 3 métodos públicos que são o *handleRequest*, *handleData* e o *refreshGraph*. O método *handleRequest* é responsável por receber os pedidos e por começar todo o procedimento lógico que culminará na geração de uma nova página para ser visualizada no *browser*. O método *handleData* é responsável por receber e interpretar os dados recebidos da camada de dados. Quanto ao método *refreshGraph*, serve para

verificar se há atualizações na rede de *players* e caso hajam são calculadas novamente as estatísticas da rede. Na vista, apenas há um método público, que é o *renderView*. A função deste método é gerar no *browser* a página das estatísticas gerais da rede de *players*, de acordo com o modelo recebido. O modelo é apenas constituído por atributos, uma vez que as suas instâncias vão ser objetos que apenas encapsulam/guardam informação. Esta informação é relativa ao número total de *players* na rede e o número de *players* em cada estado. A classe *Bluff* é uma biblioteca[1] JavaScript que permite criar gráficos em páginas web. É uma biblioteca bastante versátil e com muitas funcionalidades, pelo que no diagrama apenas estão especificados os métodos e atributos utilizados.

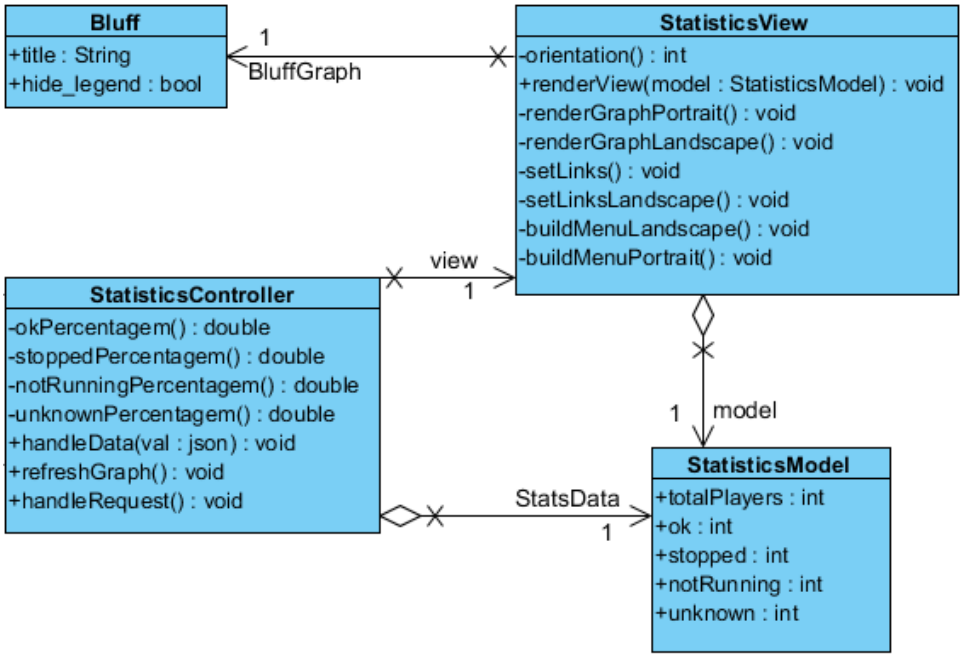


Figura 5.2: *Diagrama de classes - Statistics*

Observemos agora a figura 5.3, onde estão especificadas as classes *PlayerListController*, *PlayerListView*, *PlayerListModel* e *PlayerModel*, que são responsáveis pela página onde são listados os *players*. Temos aqui também presente o modelo MVC, onde o controlador é a classe *PlayerListController*, a vista é a classe *PlayerListView* e temos dois modelos que são as classes *PlayerListModel* e *PlayerModel*. No controlador, o método *handleRequest* é responsável por receber os pedidos e por começar todo o procedimento lógico que fará com que a página com a lista de *players* seja visualizada no *browser*. Este método recebe como parâmetro um filtro que é aplicado à lista, filtro esse que pode não conter nenhuns campos de modo a não alterar a lista

[1] http://bluff.jcoglan.com/

original. O método *getPlayers* tem como função pedir à camada de dados as informações necessárias. O método *handleData* é responsável por receber e interpretar os dados recebidos da camada de dados. Os métodos *previousPage* e *nextPage* estão encarregues da parte lógica associada à paginação da lista de *players*. O método *togglePlayerSort* tem a função de comutar a ordem pela qual a lista está ordenada. Na vista, apenas há um método público, que é o *renderView*, tendo como função gerar a página da listagem dos *players* para ser visualizada no *browser*, de acordo com o modelo recebido. Em relação ao modelo *PlayerListModel*, este é apenas constituído por atributos, que caracterizam os vários aspetos da lista de *players*. Temos também um atributo que é a própria lista dos *players*, designado por *players*, que é composta por um conjunto de objetos do modelo *PlayerModel*. Quanto ao modelo *PlayerModel*, este contém os atributos que caracterizam e definem um *player*.

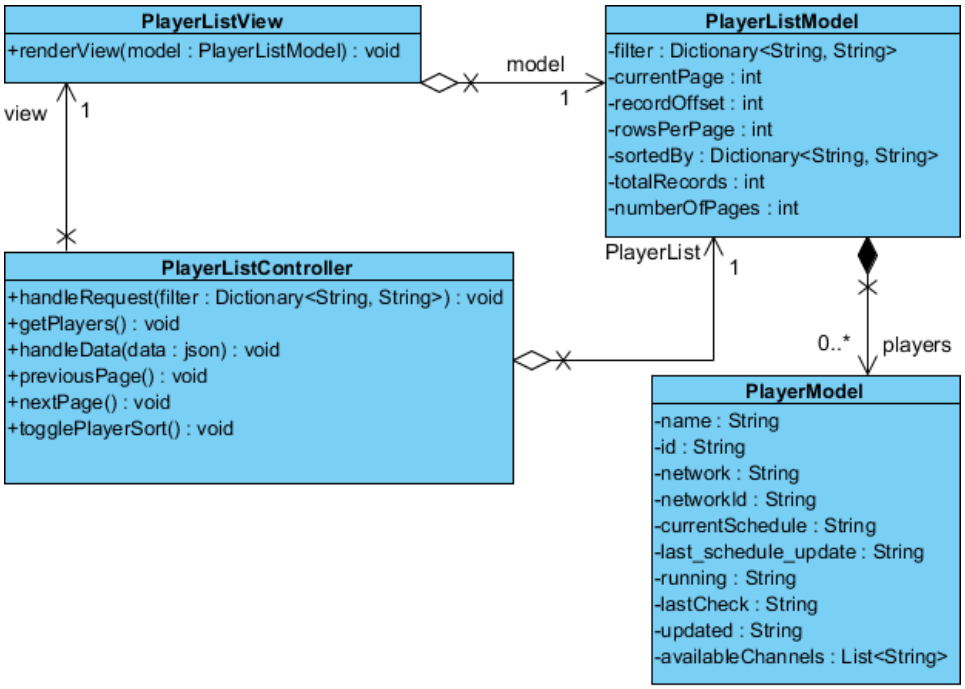


Figura 5.3: *Diagrama de classes - Player List*

Continuando a análise, temos na figura 5.4 as classes *PlayerController*, *PlayerView* e *PlayerModel*. Estas classes são responsáveis pela visualização dos detalhes de um *player*. Temos aqui também presente o modelo MVC, onde o controlador é a classe *PlayerController*, a vista é a classe *PlayerView* e o modelo é a classe *PlayerModel*. No controlador, o método *handleRequest* é responsável por receber os pedidos e por começar todo o procedimento lógico necessário para a visualização da página. O método *handleChannelData* é responsável por receber e interpretar os dados recebidos da camada de da-

dos, correspondentes à lista de canais que o *player* a ser visualizado pode sintonizar. Na vista, o método *renderView* tem como função gerar a página com os detalhes do *player* para ser depois visualizada no *browser*, de acordo com o modelo recebido. O método *updateChannels* recebe como argumento uma *hashTable* com o conjunto de canais e tem como função atualizar a página com esta nova informação. Quanto ao modelo, este já foi anteriormente analisado.

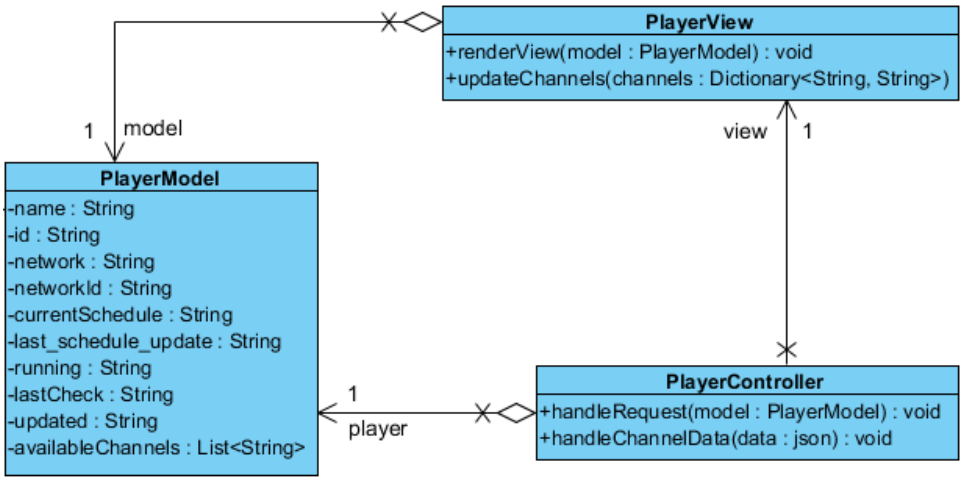


Figura 5.4: *Diagrama de classes - Player*

Finalmente temos na figura 5.5 as classes *LogsListController*, *LogsListView*, *LogListModel* e *LogModel*. São responsáveis pela visualização dos logs de um determinado *player*. Mais uma vez está presente o modelo MVC, onde o controlador é a classe *LogsListController*, a vista é a classe *LogsListView* e temos dois modelos que são as classes *LogListModel* e *LogModel*. No controlador, o método *handleRequest* é responsável por receber os pedidos e por começar todo o procedimento lógico que fará com que a página com os logs de um *player* seja visualizada no *browser*. Este método recebe como parâmetro o nome do *player* em questão e a sua identificação. O método *getLogs* tem como função pedir à camada de dados as informações necessárias. O método *handleData* é responsável por receber e interpretar os dados recebidos da camada de dados. Os métodos *previousPage* e *nextPage* estão encarregues da parte lógica associada à paginação da lista de *players*. O método *toggleSort* tem a função de comutar a ordem pela qual a lista dos logs está ordenada. Na vista, o método *renderView* tem como função gerar a página com os logs de um *player* para ser visualizada no *browser*, de acordo com o modelo recebido. O método *renderLog* vai gerar na página um género de *pop-up* com as informações completas do modelo do log que lhe é passado como parâmetro, enquanto que o método *closeLogPopUp* vai eliminar o *pop-up* que foi gerado, voltando a ser visualizada a página dos logs. Em relação ao modelo

LogListModel, este é apenas constituído por atributos, que caracterizam a lista dos logs de um *player*. Temos também um atributo que é a própria lista dos logs, designado por *logs*, que é composta por um conjunto de objetos do modelo *LogModel*. Quanto ao modelo *LogModel*, este contém os atributos que caracterizam e definem uma entrada na lista de logs, ou simplesmente um log.

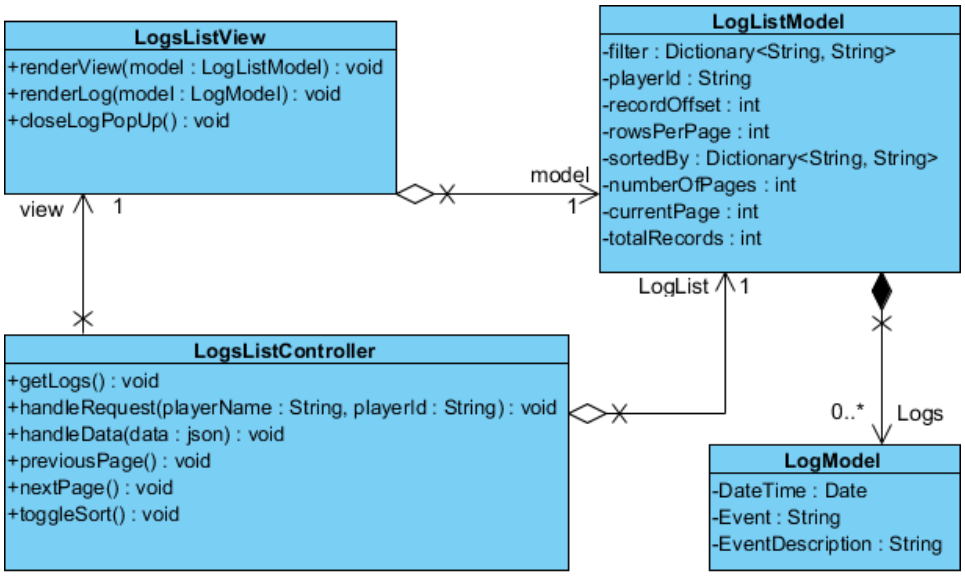


Figura 5.5: *Diagrama de classes - Logs List*

As únicas classes que faltam analisar (Ubisign.Monitoring.Proxy, RegExp, WebServices), irão sê-lo na próxima secção, uma vez que fazem parte da camada de dados.

De seguida vão ser analisados os diagramas de sequência, que especificam formalmente as várias funcionalidades da aplicação. Vão ser descritas as interações entre os objetos das várias camadas através das mensagens que são trocadas entre eles. É também perceptível a ordem pela qual as mensagens são trocadas, ou seja é feita a especificação da sequência temporal das várias interações.

O primeiro diagrama de sequência a analisar diz respeito à visualização das estatísticas sobre o estado geral da rede de *players* e pode ser observado na figura 5.6. A ação inicia-se com o utilizador a aceder ao site ou a clicar no botão de refresh que existe no topo da página. Independentemente da ação inicial do utilizador, é invocado o método *handleRequest* no controlador *StatisticsController*. De seguida, o controlador começa por criar uma instância do modelo de dados designado por *statsData* do tipo *StatisticsModel*, que vai servir para guardar os dados estatísticos calculados. Depois, o controlador, faz um pedido à camada dos dados para que esta lhe devolva a lista dos *players*. Este pedido ocorre quando se chama o método *getPlayersWS(...)* à

classe *WebServices*. Este processo de consulta dos dados é assíncrono, pelo que é necessário indicar a função (*handleData*) que deve ser chamada quando a camada de dados responder. Depois, quando a função *handleData* é chamada os dados são processados e as estatísticas calculadas. Começa-se por extrair e guardar o número total de *players* no modelo *statsData*. Depois, são percorridos os dados recebidos e para cada *player* é verificado o seu estado e é feito o update no modelo. Terminado o cálculo, é então invocada a função *renderView* da camada da interface e é passado como parâmetro o modelo *statsData* para que seja possível construir a página. Para gerar a página, começa-se por verificar qual é a orientação do dispositivo, uma vez que a página vai ser organizada de maneira diferente consoante a orientação. Depois da página ser gerada de acordo com a orientação do dispositivo, é renderizada no *browser*.

O próximo diagrama a ser analisado diz respeito à visualização da lista de *players*, e pode ser observado na figura 5.7. Podemos ver que no início do diagrama há uma referência para a visualização das estatísticas do estado geral da rede de *players*. Isto deve-se ao facto de ser a partir dessa página que se consulta a lista de *players*, através dos vários links aí presentes. Portanto, o utilizador inicia a ação ao clicar num dos cinco links da página principal (Homepage). Estes links servem para de uma forma rápida se consultar a lista de *players* de acordo com o seu estado ("playing","stopped","not running","unknown"). Esta ação está representada no frame *alt* que define cinco regiões mutuamente exclusivas correspondentes a cada um desses links. Depois, é invocado o método *handleRequest(filter : Dictionary<String,String>)* no controlador *PlayerListController*, com o parâmetro *filter* de acordo com o link seguido pelo utilizador. De seguida, o controlador começa por criar uma instância do modelo de dados designado por *playerList* do tipo *PlayerListModel*, que vai servir para guardar a lista de *players* e outras informações auxiliares. Depois, o controlador acede à camada de dados para pedir uma porção da lista de *players*, e não a lista completa porque a sua visualização é paginada. Mais uma vez, o processo de acesso aos dados é assíncrono, pelo que é necessário indicar a função (*handleData*) que deve ser chamada quando a camada de dados responder. Depois, quando a função *handleData* é chamada, o controlador vai começar o processo de extração da informação e construção do modelo de dados. Começa por criar uma instâcia da classe *RegExp*, que vai servir para extrair a informação relevante dos dados recebidos. De seguida é atualizado no modelo o número total de *players*. O próximo passo consiste em guardar no modelo *playerList* a lista de *players*. Para isso é percorrida a lista de *players* dos dados recebidos e para cada um extrair a informação relevante, criar uma instância de *playerModel* com essa informação e adicioná-la na lista do modelo. Terminada a parte lógica no controlador, é então invocada o método *renderView* da camada da interface sendo passado como parâmetro o modelo de dados. Na camada da interface, é a classe *PlayerListView* que fica encarregue de gerar a página de acordo

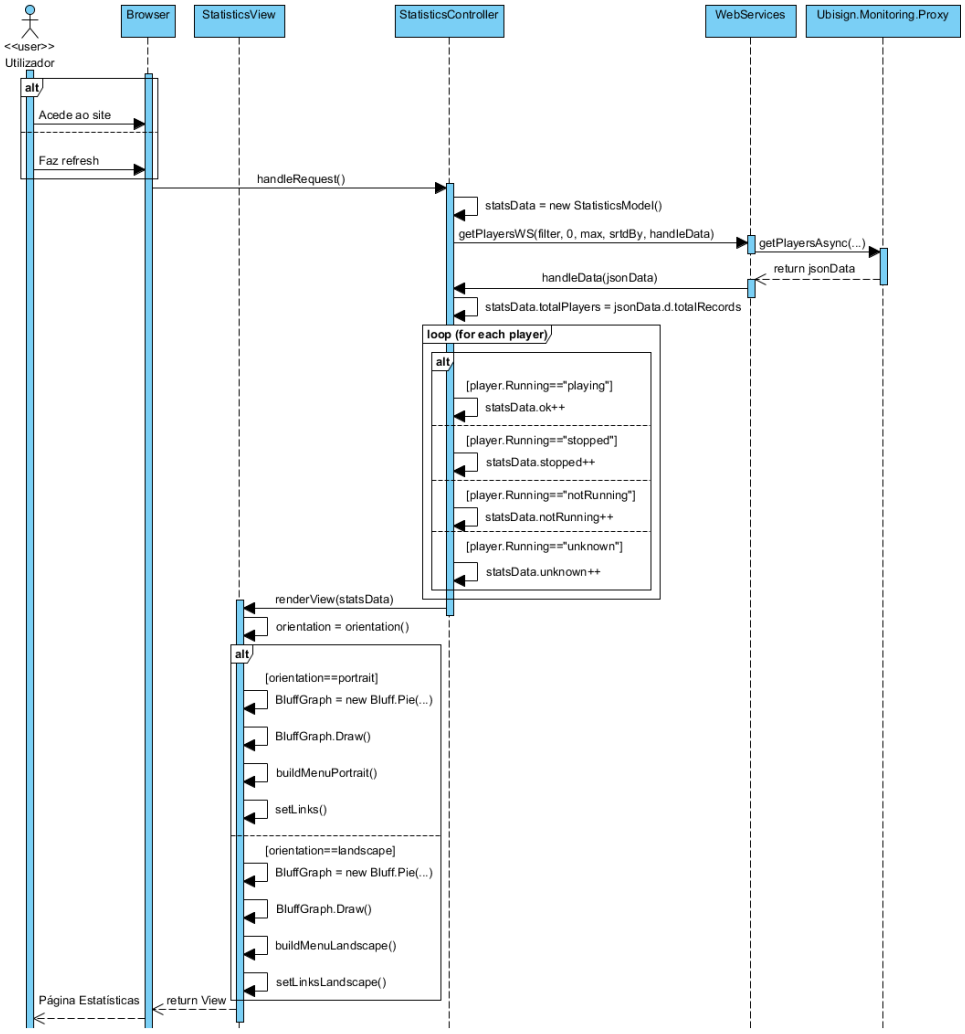


Figura 5.6: *Diagrama de sequência: visualizar as estatísticas sobre o estado geral da rede de* players.

com o modelo recebido para depois ser visualizada no *browser*.

O próximo diagrama a ser analisado diz respeito à visualização dos detalhes de um *player*, e pode ser observado na figura 5.8. No início do diagrama há uma referência à listagem dos *players*, uma vez que é a partir desta página que o utilizador escolhe um *player* para ver os seus detalhes/informações. Portanto, a ação inicia-se quando o utilizador seleciona um *player* da lista que está a visualizar. Em resposta a esta ação, é invocado o método *handleRequest(model)* no controlador *PlayerController*. O modelo passado como parâmetro é do tipo *PlayerModel* e contém a informação completa do *player* selecionado. Isto é possível uma vez que na visualização da lista dos *players* já se tem a informação completa de cada um dos *players* guardada.

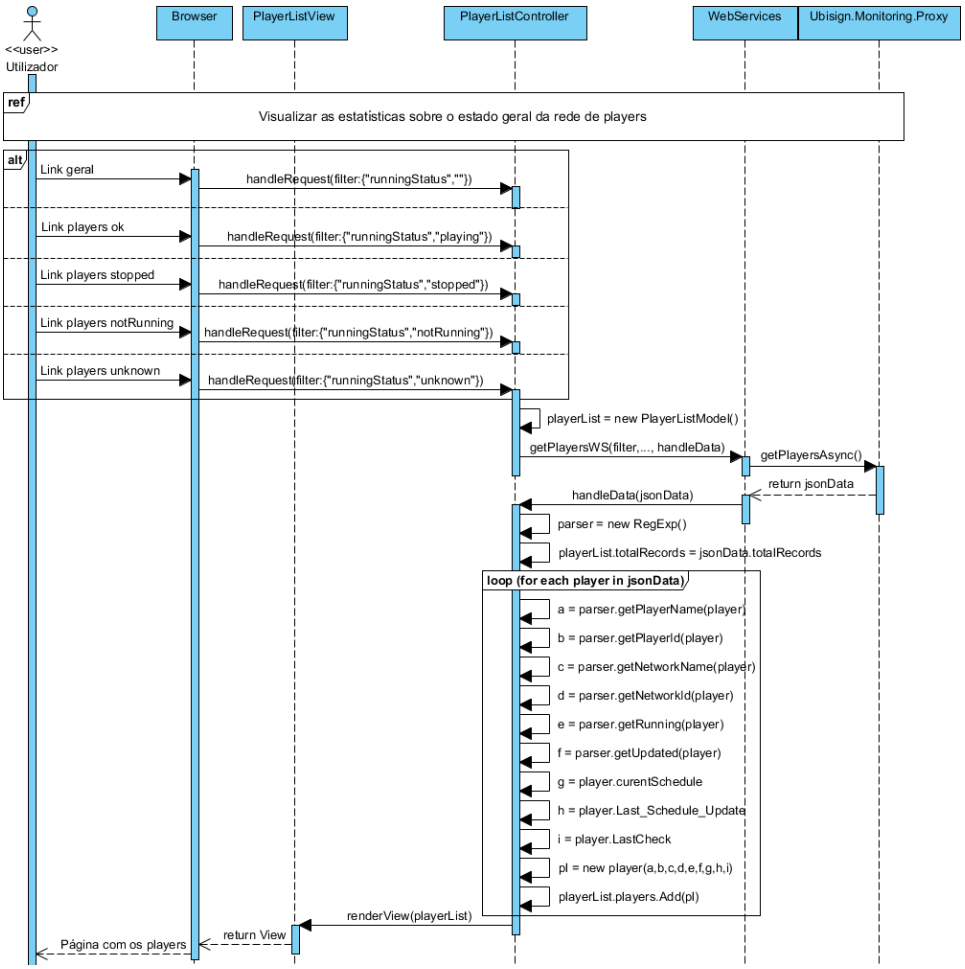


Figura 5.7: *Diagrama de sequência: ver a listagem dos players.*

Desta forma não é necessário aceder à camada de dados e pedir a informação desse *player*, e por isso é logo invocado o método *renderView(model)* à vista que é uma instância da classe *PlayerView*. Depois a página é gerada de acordo com o modelo do *player* e visualizada no *browser*. No entanto, o controlador depois de invocar o método da vista ainda tem mais uma tarefa. É necessário aceder à camada de dados para consultar a lista de canais disponíveis para aquele *player*. Desta forma, é invocado o método *getNetworkChannelsWS(nID, handleChannelData)* à camada de dados, onde o parâmetro *nID* é a identificação da rede do *player* e o outro parâmetro é a função que deve ser chamada quando houver resposta da camada de dados, uma vez que é uma operação assíncrona. Quando se obtém a resposta, o controlador apenas vai validar os dados recebidos na função *handleChannelData* e vai chamar o método *updateChannels* da vista. Esta função, que recebe a lista de canais no formato JSON (JavaScript Object Notation) como parâ-

metro, vai atualizar a página com os canais, sendo que agora o utilizador já poderia mudar o canal sintonizado do *player*.

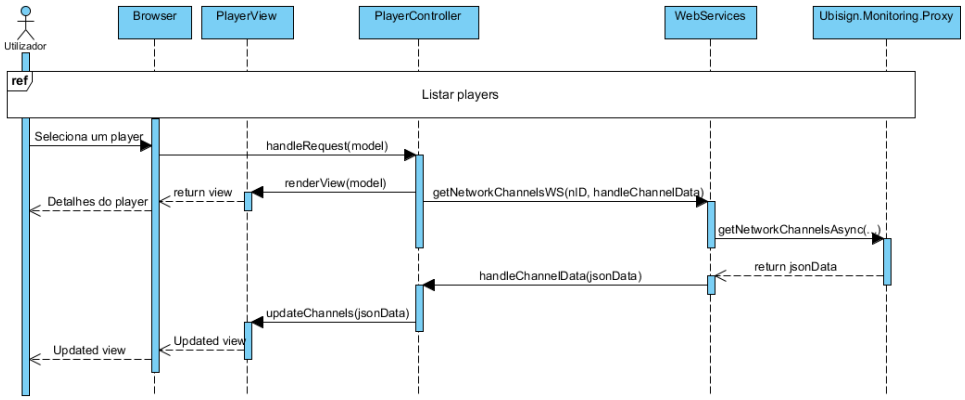


Figura 5.8: *Diagrama de sequência: ver detalhes de um player.*

Finalmente, o quarto diagrama de sequência a ser analisado diz respeito à visualização dos logs de um *player* e pode ser observado na figura 5.9. Podemos ver que no início do diagrama há uma referência para a visualização dos detalhes de um *player*. Isto deve-se ao facto de ser a partir dessa página que se consultam os logs desse *player*. Portanto, a ação inicia-se quando o utilizador clica no link dos logs presente na página. Esta ação vai dar início a todo o processo, que começa com a invocação do método *handleRequest(playerName, playerID)* no controlador *LogsListController*. O parâmetro *playerName* é o nome do *player* e o *playerID* é a identificação do *player*, que é necessária para depois efetuar o pedido dos logs à camada de dados. O controlador começa por criar uma instância do modelo de dados designada por *logList* do tipo *LogListModel*, que vai servir para guardar os logs e outras informações auxiliares. Depois, o controlador acede à camada de dados para pedir os logs que estão associados ao *player* cuja identificação é passada como parâmetro. Como o processo de acesso aos dados é assíncrono, é indicada a função *handleData* que deve ser chamada quando houver uma resposta com os dados pedidos. Depois, o método *handleData* é invocado no controlador com os dados que foram recebidos e vão ser copiados para o modelo *logList*. Começa-se por guardar o número total de registos no modelo e depois é percorrida a estrutura de dados recebida e para cada entrada são extraídas as informações necessárias e salvaguardadas no modelo. Terminada a operação da construção do modelo, é então chamado o método *renderView(logList)* da camada da interface, que irá gerar a página de acordo com o modelo passado como parâmetro.

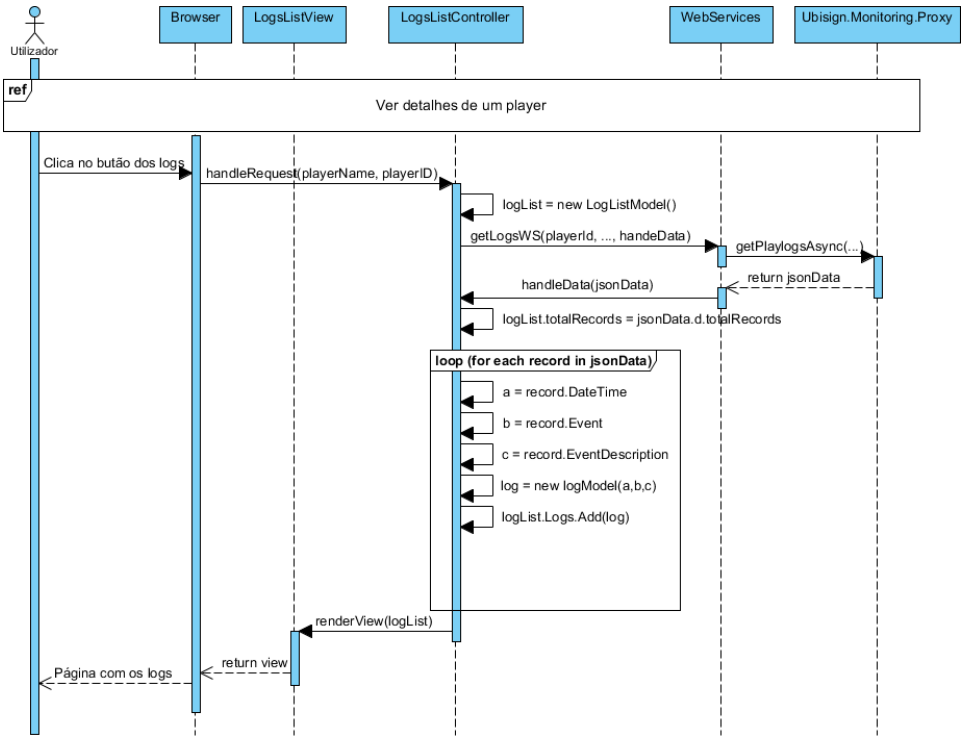


Figura 5.9: *Diagrama de sequência: consultar os logs de um player.*

5.2 Camada de dados

A camada de dados desta aplicação ficou bastante simplificada devido à impossibilidade de se poder aceder e manipular diretamente os dados no lado do servidor da aplicação. Desta forma, ficou estabelecido que o acesso aos dados seria feito através de pedidos por *web services*. Analisando os requisitos funcionais da aplicação, a camada de dados teria que suportar as 4 operações seguintes:

1. Consultar a lista dos *players*.
2. Consultar a lista de canais disponiveis numa determinada rede.
3. Alterar o canal sintonizado de um *player*.
4. Consultar os logs de um determinado *player*.

Contudo, estas 4 operações têm que ser parametrizáveis e dinâmicas de modo a dar resposta às necessidades da aplicação. Na operação de consulta da lista dos *players* deve ser possível especificar e aplicar os seguintes critérios:

- Aplicar um filtro à lista de acordo com vários parâmetros, nomeadamente o estado dos *players* e o estado do canal.

- Especificar uma determinada porção da lista de *players* que se quer consultar, uma vez que a visualização dessa lista é paginada e, portanto, apenas se quer a informação estritamente necessária.
- Especificar a ordenação da lista por ordem crescente ou decrescente de acordo com o nome dos *players*.

Por sua vez, na operação de consulta dos logs de um determinado *player* deve ser possível especificar e aplicar os seguintes critérios:

- Aplicar um filtro à lista de acordo com vários parâmetros, nomeadamente um intervalo horário e um intervalo temporal.
- Especificar uma determinada porção da lista dos que se quer consultar, uma vez que a visualização dessa lista é paginada e, portanto, apenas se quer a informação estritamente necessária.
- Especificar a ordenação da lista por ordem crescente ou decrescente de acordo com a data dos eventos presentes na lista.

Quanto às operações 2 e 3, estas já são mais simples e portanto apenas precisam de certos identificadores para processar o pedido. A operação de alterar o canal sintonizado de um *player* apenas precisa do identificador do *player* e do identificador do canal escolhido. Por sua vez a operação de consultar a lista de canais disponíveis numa determinada rede, apenas necessita do identificador da rede, uma vez que nesta operação não faz sentido aplicar qualquer tipo de filtragem ou ordenação. Na figura 5.10 podem ser observadas as duas classes que compõem a camada de acesso aos dados e que refletem as especificações acabadas de mencionar.

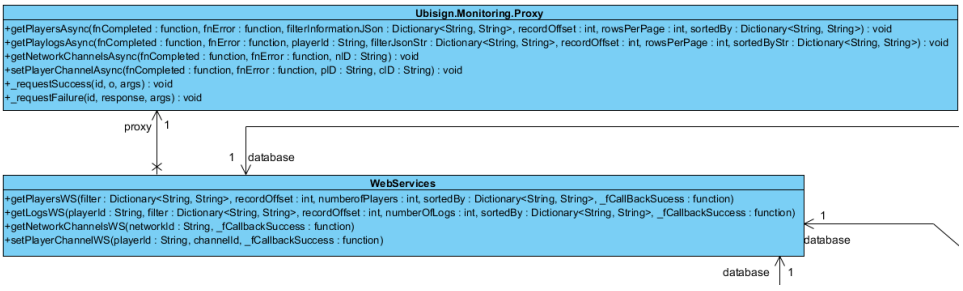


Figura 5.10: *Diagrama de classes: Camada de dados*

Um problema que surge com o acesso indireto aos dados através de web services, é que não temos controlo sobre a formatação e estruturação dos dados recebidos. Os dados são recebidos no formato JSON, o que à partida não cria nenhum problema pois é um formato standard e é facilmente interpretado pelo javascript. Esses dados são interpretados como uma tabela

de *hash* onde temos pares chave-informação do tipo *String-String*. O problema que se levanta tem a ver com a string da informação, uma vez que tem muita informação que não é relevante, e portanto há a necessidade de extrair apenas o conteúdo útil para a aplicação. Desta forma foi necessário criar mecanismos de validação e extração da informação relevante. Foi então criada a classe *RegExp*, que consiste em vários métodos que fazem o *parsing* da informação, de acordo com a estruturação usada pela Ubisign, usando para isso expressões regulares. Para além de extrair a informação relevante, garante também que os dados recebidos são válidos, ou seja, estão estruturados de acordo com a especificação. Na figura 5.11 pode ver-se a especificação da classe em questão. Todos os métodos recebem por parâmetro uma string, que é a informação que foi recebida, e retornam uma string que apenas contém a informação pretendida. Importa também dizer que, caso ocorra alguma excepção no *parsing* devido aos dados não respeitarem a estruturação, é retornada uma *string null*.

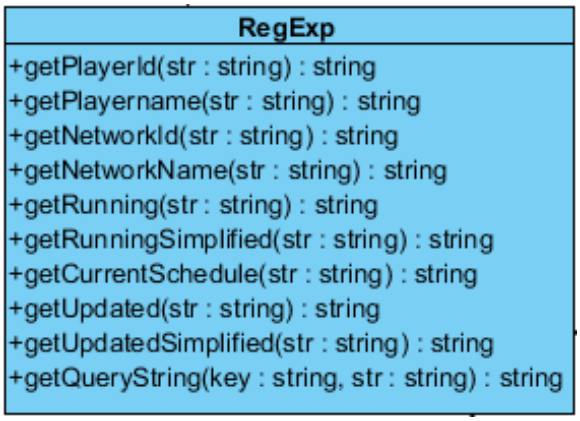


Figura 5.11: *Diagrama de classes: Parser*

5.3 Processo de desenvolvimento

Nesta secção vão ser abordados os aspetos gerais e a forma como decorreu todo o processo de desenvolvimento.

De um modo geral pode dizer-se que o processo de desenvolvimento foi baseado no modelo em cascata, embora com algumas modificações. Na figura 5.12 pode-se observar um diagrama geral do processo de desenvolvimento onde estão representadas as suas etapas principais, que serão detalhadas no decorrer desta secção. Pelo diagrama podemos observar que é um modelo de desenvolvimento sequencial em que as etapas estão dependentes umas das outras, embora se possa retroceder para melhorar ou implementar novas funcionalidades numa etapa anterior.

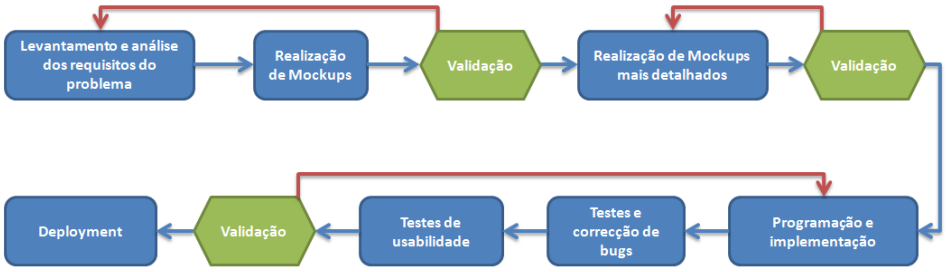


Figura 5.12: *Diagrama do processo de desenvolvimento*

Como se pode constatar pela figura 5.12 o processo de desenvolvimento teve início com o levantamento e análise dos requisitos do problema através de várias reuniões na empresa. Para que houvesse uma boa compreensão geral do problema foi em primeiro lugar feita uma explicação detalhada da plataforma e serviços atualmente existentes na empresa. Para a organização e documentação dos requisitos foi utilizado o programa *Visual Paradigm for UML*. Toda a documentação e diagramas associados ao levantamento e análise dos requisitos do problema podem ser consultados em anexo.

De seguida foram feitos os *mockups* de baixo detalhe, pouco detalhados e com realce para os requisitos mais importantes e prioritários. O objetivo principal da realização de *mockups* de baixo detalhe é poder rapidamente fazer um esboço geral da aplicação e validar com o cliente os seus traços gerais para assim poder numa fase inicial obter o feedback e ajustar certos parâmetros. Pode-se também constatar que os requisitos não mudaram nem foram acrescentados outros a meio do projeto, fruto de uma boa fase inicial e uma rápida validação. Nas figuras 4.2 e 4.3 podemos ver os *mockups* de baixo detalhe realizados nesta fase.

Na etapa seguinte, foram realizados *mockups* com um nível de detalhe muito próximo ao que iria ser implementado. Podemos ver pelas figuras 4.4 e 4.5 que o *layout* da etapa anterior foi modificado. Essas alterações tiveram origem no processo de validação dos *mockups* de baixo detalhe, pois não iam de encontro com as espectativas e visão inicial do cliente. Nesta fase, e à medida que foram feitos os *mockups* também foram feitos os Use Case textuais UML, que descreviam ao promenor o modo de funcionamento e interação com cada página. Estes Use Cases textuais podem ser consultados nos anexos. Tanto nesta fase como na anterior foi utilizado o programa *Mockflow* para fazer os protótipos e os *mockups*.

Uma vez validados os *mockups* de alto detalhe, iniciou-se a fase da programação e implementação. Esta fase começou com a especificação UML de todas as classes, atributos e métodos que iriam compôr cada uma das camadas da aplicação a desenvolver. Mais uma vez, toda a documentação UML relacionada com esta fase pode ser consultada em anexo.

Na fase seguinte, testes e correção de bugs, foram realizados vários tipos de testes de modo a garantir o correto funcionamento da aplicação. Depois, nos testes de usabilidade, a aplicação foi testada em vários *browsers* e sistemas operativos de *smartphones* de modo a testar a sua compatibilidade.

Após a fase da programação e dos vários testes, foi feita uma validação no cliente de modo confirmar e validar que a aplicação estava implementada de acordo com o que tinha sido pedido.

O deployment e integração da aplicação nos serviços do cliente foi feito por pessoas na empresa, uma vez que eu não tenho acesso à parte de administração do sistema informático. Posso somente dizer que a integração foi bem sucedida e que a aplicação está a correr conforme previsto.

O tempo alocado a cada uma das etapas do processo de desenvolvimento pode ser observado no diagrama de Gantt representado na fig. 5.13. Nesse diagrama estão também representadas as dependências entre as etapas e as datas do início e fim de cada etapa.

	Task Name	Duration	Start	Finish	Predecessors
1	Levantamento e análise dos requisitos	39 days	Mon 25-10-10	Thu 16-12-10	
2	Mockups de alto nível	7 days	Fri 05-11-10	Mon 15-11-10	
3	Validação	3 days	Tue 16-11-10	Thu 18-11-10	2
4	Mockups de baixo nível	15 days	Fri 19-11-10	Thu 09-12-10	3
5	Validação	5 days	Fri 10-12-10	Thu 16-12-10	4
6	Programação e implementação	60 days	Fri 17-12-10	Thu 10-03-11	5;1
7	Testes e correcção de bugs	15 days	Fri 11-03-11	Thu 31-03-11	6
8	Testes de usabilidade	15 days	Mon 21-03-11	Fri 08-04-11	6
9	Validação Final	20 days	Mon 11-04-11	Fri 06-05-11	8;7
10	Deployment	19 days	Mon 09-05-11	Thu 02-06-11	9

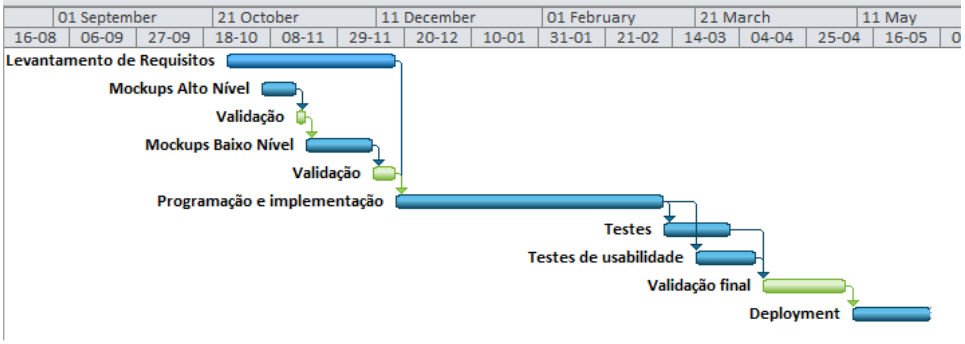


Figura 5.13: *Planeamento do projeto*

Na fig. 5.14 podemos observar melhor o tempo, em percentagem, alocado a cada etapa. As duas etapas principais foram as que ocuparam a maior parte do tempo. São elas a etapa de levantamento e análise de requisitos e a etapa da programação e implementação, que ocuparam 50% do tempo total. As duas etapas de desenvolvimento de *mockups* ocuparam no seu conjunto 12%. As etapas de validação foram também bastante importantes e ocuparam 14% do tempo. As etapas de testes de bugs e de usabilidade ocuparam no

seu conjunto cerca de 16%, o que é uma percentagem de tempo bastante razoável para as tarefas em questão.

A etapa de deployment acabou por ocupar 10% do tempo total de desenvolvimento, o que à primeira vista pode parecer um pouco exagerado. Mas isto deve-se ao facto da etapa de deployment incluir também um certo período de tempo para testar a aplicação em cenários de uso real para ver se realmente tem um comportamento estável e dentro do esperado.

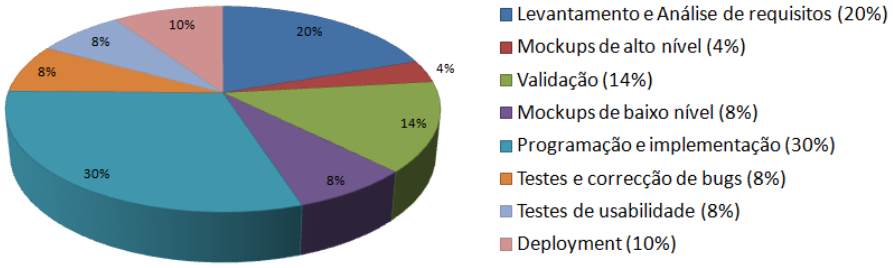


Figura 5.14: *Percentagem de tempo alocado a cada uma das etapas*

5.4 Solução final

Nesta secção vamos analisar a versão final da aplicação desenvolvida. A análise vai centrar-se na usabilidade e no aspeto visual da aplicação, não sendo abordados pormenores técnicos ao nível da implementação. As imagens da aplicação final foram capturadas num contexto de uso real usando o serviço da Ubisign. Para poder ter acesso aos serviços de administração e gestão dos players, foi preciso criar e registar uma conta no serviço da Ubisign. Naturalmente, por razões de confidencialidade e segurança por parte da Ubisign para com os seus clientes, a conta criada não teria acesso à rede geral de players. Posto isto, para simular um cenário de uso real foram criadas 32 máquinas virtuais com o serviço de digital signage da ubisign, que formam o conjunto de players do sistema. Cada um dos players esteve algum tempo ativo para gerar entradas na lista de logs.

Comecemos por analisar a página do login, representada na figura 5.15. Sobre esta página há pouco a dizer, é minimalista e vai de encontro ao que foi projetado nos mockups iniciais. As mensagens de erro são bem visíveis e sucintas para que o utilizador se aperceba do erro.

Vamos analisar agora para a página inicial, ou *homepage*, que está presente na figura 5.16. Aqui podemos ver a página com dois tipos de *layout*

Figura 5.15: *Solução final: Login*

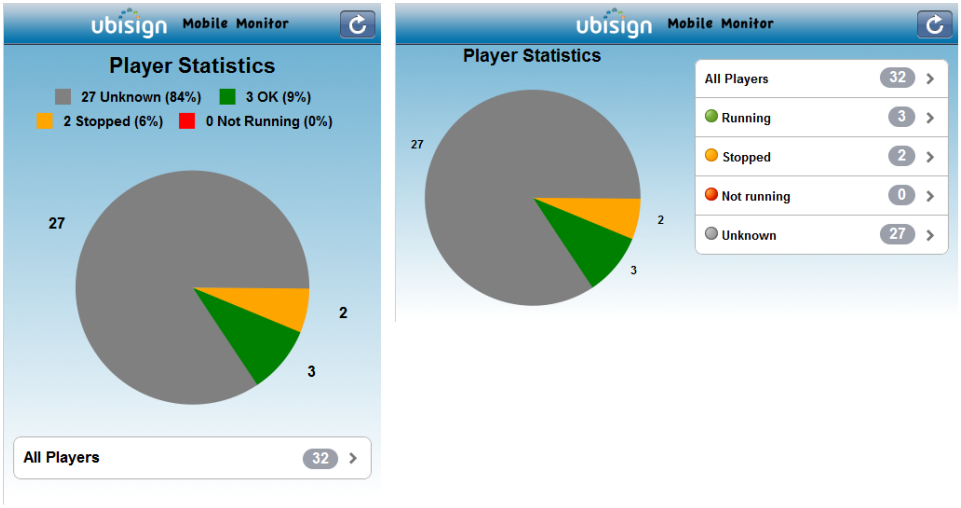


Figura 5.16: *Solução final: Homepage*

dependendo da orientação do *smartphone*. A página da esquerda é visualizada quando o *smartphone* está na posição *portrait* e a página da direita corresponde à posição de *landscape*. A comutação entre estes dois *layouts* é automática e completamente transparente ao utilizador. A razão para utilizar estes dois *layouts* diferentes prende-se com a melhoria da experiência de utilização que se verifica com a reorganização e adaptação da informação à orientação do ecrã. De resto, a informação presente nesta página vai de encontro aos requisitos, que pediam um gráfico *pie chart* com o estado geral da rede indicando também o número de players em cada estado. A barra de navegação, onde está presente o símbolo da Ubisign e o nome da aplicação, vai servir para facilitar o utilizador a navegar pela aplicação, nomeadamente voltar para a página anterior ou ir diretamente para a página inicial. Como estamos na página inicial e para a barra de navegação não ficar sem utilidade aproveitou-se para adicionar um botão que atualiza o gráfico.

Seguindo os vários links presentes na página inicial, o utilizador vai para a página com a listagem dos players, que pode ser observada na figura 5.17. Comecemos por analisar a barra de navegação da aplicação, que vai permanecer igual para as restantes páginas da aplicação. Aqui podemos encontrar à esquerda um botão que faz com que a aplicação volte para a página anterior e à direita um que redireciona o utilizador para a página inicial (Homepage). Logo a seguir à barra da navegação temos o mecanismo para filtrar a lista de players. Depois vem a lista dos players propriamente dita. A navegação entre as várias páginas da lista é feita pelos dois botões *Prev* e *Next* ou através de um movimento *swipe* (parecido ao folhear de uma página) para a esquerda ou para a direita consoante se queira ir para a página anterior ou seguinte. Ao navegar pela lista, sempre que o utilizador for para uma página nova é

Figura 5.17: *Solução final: Listagem dos Players*

dado o feedback ao utilizador que a página está a ser carregada, como se pode ver na imagem da direita na figura 5.17. Na lista, para cada player era necessário poder visualizar as seguintes informações: nome do player, estado do player, nome da rede a que pertence, nome do canal sintonizado e o estado do canal. De certa forma é muita informação para ser visualizada numa interface tão pequena, sem que se fique com uma página ilegível. A solução encontrada para ao mesmo tempo se poder visualizar toda essa informação e ter uma interface limpa e agradável, foi agregar o nome do player e o seu estado bem como o nome do canal sintonizado e o seu estado. Deste modo, temos o nome do player seguido de um simbolo que identifica o seu estado. De forma idêntica o nome do canal tem à sua frente um símbolo com o seu estado. No contexto da aplicação, pode-se tomar essa liberdade uma vez que os utilizadores da aplicação já estão familiarizados com o sistema e código de cores e portanto esta abordagem não interfere com a compreensão da informação. Outro pormenor da lista é a coloração alternada das linhas, que visualmente fica mais apelativo e menos confuso.

A partir da listagem dos players, pode-se selecionar da lista um player de modo a podermos visualizar os seus detalhes completos. Esta página pode ser observada na figura 5.18. Também aqui, de um modo similar à página inicial, temos dois tipos de *layout* dependendo da orientação do *smartphone*. A página da esquerda é visualizada quando o *smartphone* está na posição *portrait* e a página da direita corresponde à posição de *landscape*. A comutação entre estes dois *layouts* é automática e completamente transparente ao utilizador. No entanto, o modo como se deteta a orientação e se constrói a página com o *layout* adequado é diferente nos dois casos. Na página inicial, a orientação

Figura 5.18: *Solução final: Detalhes de um Player*

é detetada pelo javascript e em função disso a vista gera o *layout* correto. No caso desta página, os dois *layouts* são conseguidos usando uma funcionalidade do CSS3 que permite especificar os estilos a aplicar aos elementos da página consoante esta esteja no modo *portrait* ou *landscape*. Desta forma é mais simples, mas tal só é possível porque o conteúdo da página é o mesmo nos dois *layouts* havendo apenas uma reorganização dos vários elementos, ao contrário do que acontece na página inicial que há alteração de vários elementos. Em relação às funcionalidades presentes nesta página, temos a opção de modificar o canal sintonizado pelo player através de uma dropbox e temos também a possibilidade de visualizar os logs clicando no símbolo para o efeito.

Vamos agora analisar a página dos logs de um player, que está representada na figura 5.19. Podemos ver que cada entrada na lista tem 3 campos informativos que são a data do evento, a sua classificação/nome e depois uma descrição mais detalhada sobre o evento. Como a descrição do evento na maior parte dos casos é relativamente longa, o texto é truncado de modo a não ocupar espaço em demasia. No entanto se o utilizador quiser ver a descrição completa pode fazê-lo clicando na entrada da lista e aparece um *pop-up* com os detalhes completos desse evento. Este cenário pode ser observado na imagem superior direita da figura 5.19. Em relação à paginação dos logs, é feita de uma forma idêntica à listagem dos players. O utilizador pode navegar pela lista de duas formas, pelos botões *Prev* e *Next* ou com um movimento *swipe*, semelhante ao movimento de folhear uma página. Sempre que uma página da lista está a carregar é dado o feedback ao utilizador, como se pode ver na imagem inferior esquerda da figura 5.19. Outro pormenor relativo aos eventos, é a coloração de eventos especiais. Temos por exemplo, a

coloração a verde nos logs com a descrição do evento "PlayerAction.playing" ou a vermelho quando é "PlayerAction.stopped". É uma forma de chamar a atenção do utilizador para eventos que mereçam a sua atenção por serem mais relevantes. Em relação à filtragem da lista, temos na zona inferior da página os controlos responsáveis por essa funcionalidade. De referir que os dados introduzidos no intervalo de datas, são auxiliados por um calendário que aparece de modo a ser mais prático. Na barra da navegação temos, como já foi dito, o botão para regressar à página anterior e outro para ir diretamente para a página inicial.

Como nota final, importa referir que as transições entre as páginas da aplicação ocorrem com uma animação que dá a sensação de se tratar de uma aplicação nativa. Isto é conseguido devido ao uso da framework jQtouch.

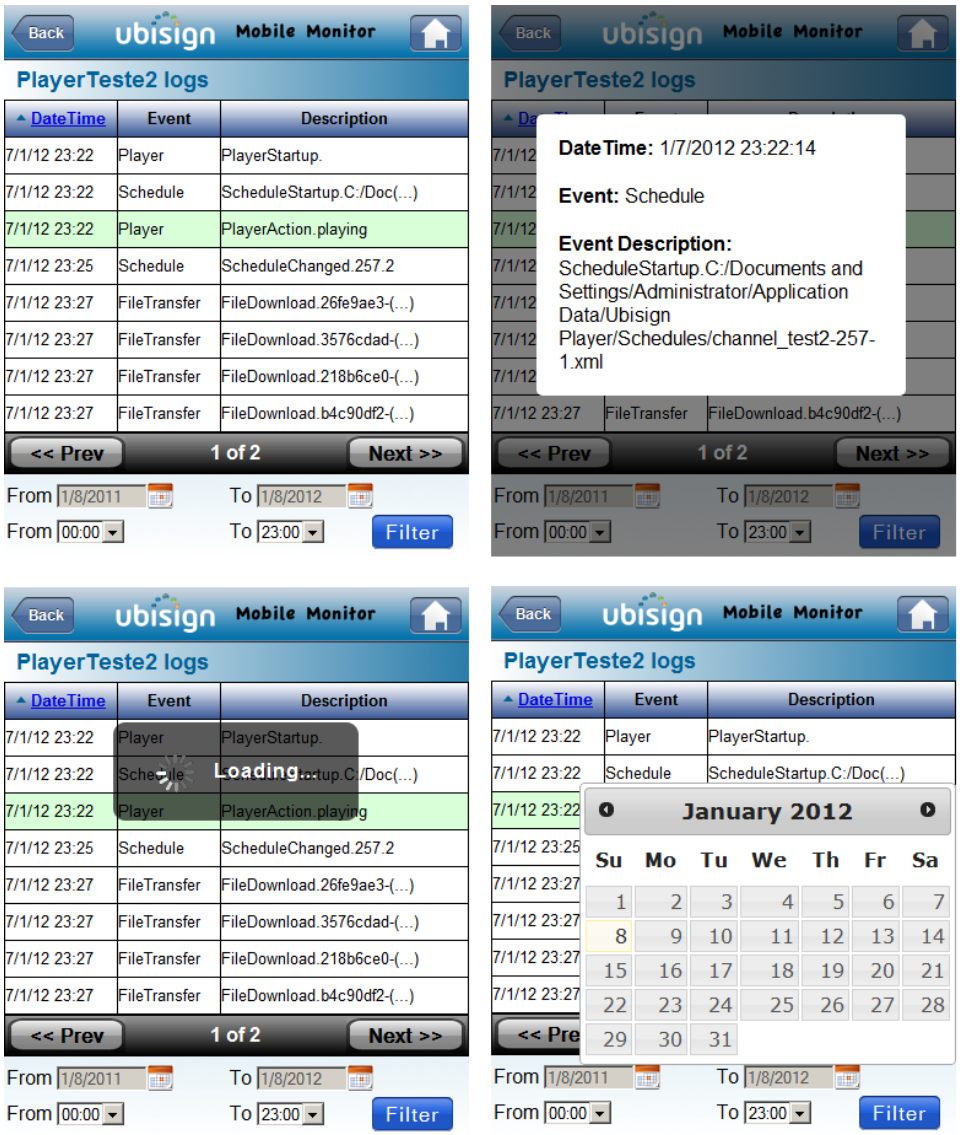


Figura 5.19: *Solução final: Visualização dos logs de um Player*

Capítulo 6

Testes

Neste capítulo vão ser analisados, de uma forma geral, os vários testes realizados à aplicação. Em primeiro lugar, vão ser apresentados os testes de validação, que têm como objetivo assegurar o correto funcionamento da aplicação quando esta manipula dados. Depois, são apresentados e analisados os testes de compatibilidade e usabilidade que têm como objetivo assegurar o correto funcionamento e visualização da aplicação em diversos *browsers*. No final, é feita uma apreciação geral dos resultados obtidos na realização dos testes.

6.1 Testes de validação

A validação foi levada a cabo com testes exaustivos que tinham como objetivo assegurar o correto funcionamento da aplicação quando esta manipula dados, quer estes sejam coerentes ou não. Estes dados podem ter várias origens tais como pedidos a serviços externos (*web services*) à aplicação, da interacção com o utilizador ou até gerados pela própria aplicação em tempo real.

6.1.1 Testes realizados

Após uma análie detalhada à estrutura do projeto, foram identificadas 10 classes que compõem as camadas da interface, da lógica de negócio e dos dados, estando representadas na tabela 6.1. Estas classes controlam grande parte da aplicação e o seu correto funcionamento é fundamental para que a aplicação funcione de uma forma correta e de acordo com as suas especificações.

Os testes de validação realizados foram divididos em dois grupos distintos, classificados como "*Inputs* válidos" e "*Inputs* inválidos". De seguida são explicados em que consistem estes dois tipos de testes:

Classe	# de funções	# funções cobertas	% funções cobertas
StatisticsView	8	8	100%
StatisticsController	7	7	100%
PlayerListView	1	1	100%
PlayerListController	6	6	100%
PlayerView	2	2	100%
PlayerController	2	2	100%
LogsListView	3	3	100%
LogListController	6	6	100%
RegExp	11	11	100%
WebServices	4	4	100%
Total	50	50	100%

Tabela 6.1: *Cobertura de código dos testes de validação*

- *Inputs* válidos - Estes testes efetuam chamadas a métodos fornecendo-lhes parâmetros/inputs válidos. Para que os testes efetuados tenham sucesso é preciso que os métodos devolvam o resultado correto e que saibam gerir qualquer exceção interna. A título de exemplo podem considerar-se os métodos da classe RegExp (Expressões Regulares), os quais perante uma *string* válida devem conseguir fazer o *parsing* e extrair a informação corretamente.

- *Inputs* inválidos - Estes testes efetuam chamadas a métodos fornecendo-lhes parâmetros/inputs inválidos. Nestes casos, a aplicação deve conseguir identificar que está perante dados inválidos e deve reagir de forma controlada para não comprometer o funcionamento global da aplicação. Os *inputs* inválidos podem manifestar-se de muitas maneiras, desde parâmetros a *null*, índices negativos ou fora do limite de um array, caracteres inválidos ou até dados XML mal estruturados.

Das dez classes testadas, duas foram alvo de um conjunto de testes mais abrangentes e exigentes por terem um papel crítico no funcionamento da aplicação, sendo elas a classe *RegExp* e a classe *WebServices*. A classe *WebServices* é responsável pelos pedidos à base de dados, sejam estes de escrita ou consulta, enquanto que a classe *RegExp* é responsável por interpretar o conjunto de dados devolvidos pela base de dados. No que diz respeito à classe que efetua o acesso à base de dados através de *web services*, os testes

tinham como objetivo testar as funções perante qualquer cenário anormal. Existem muitos fatores que podem originar erros e exceções sendo necessário que estes casos sejam tratados devidamente de modo a que a aplicação não fique comprometida. Temos, por exemplo, casos onde as ligações caem, ou se fica temporariamente sem ligação à rede, ou por qualquer razão a base de dados demora muito tempo a responder e é necesário atribuir *timeouts* a estes pedidos. Depois, pelo facto do acesso à base de dados ser feito por *web services* e estes não estarem sob o nosso controlo, é preciso assegurarmo-nos que toda a informação recebida e enviada é correta e está bem estruturada. Quanto à classe *RegExp*, os testes tinham como objetivo ver se as funções interpretam corretamente os dados recebidos pela base de dados. Serviram também para garantir que as funções, perante dados incorretos e mal estruturados, não bloqueiam nem lançam exceções que não estão previstas. Como não se tinha acesso à base de dados nem ao servidor que recebe os pedidos, foi montada uma base de dados local e também um servidor que recebia os pedidos via *web services*. Com o servidor de testes foi possível testar os vários cenários contemplados nos muitos testes realizados que de outra forma não poderiam ter sido levados em conta. Foram, por exemplo, considerados cenários de perdas de ligação ao servidor dos dados, acessos recusados pelo servidor, respostas com muito tempo de atraso, respostas incorretas na sua estrutura e conteúdos, etc.

Assim, pretende-se assegurar que a aplicação tem um funcionamento correto de acordo com as especificações, cumprindo desta forma os requisitos do projeto. Numa fase inicial vários métodos não passaram nos testes, por várias razões, maioritariamente devido a cenários específicos que não tinham sido tidos em consideração. Muitos destes casos específicos só foram detetados nos testes, porque em utilização normal seriam raros de acontecer. À medida que os métodos falhavam eram analisadas as causas e corrigidas, sendo que no final da fase de testes a aprovação foi de 100%. No entanto, isto não significa que a aplicação não tenha qualquer falha, mas sim que, face aos testes realizados e que foram bastante abrangentes, todas as falhas detetadas foram corrigidas.

Nas tabelas 6.2 e 6.3 são apresentados os resultados finais dos testes realizados. Como era de esperar, no final todos os métodos passaram com sucesso nos vários testes. Importa mais uma vez ressalvar que a aplicação não é infalível, mas sim que face à abrangência dos testes é uma aplicação robusta.

6.1.2 Cobertura de código

Por cobertura de código deve entender-se a quantidade de blocos de código que foram testados nos vários testes efetuados. Logicamente, uma maior cobertura significa que há uma maior percentagem de código abrangido pelos testes. No entanto, tendo em conta a arquitetura da aplicação, não é pos-

Classe.Função	Resultado	Erro
StatisticsView.orientation	Passed	
StatisticsView.renderView	Passed	
StatisticsView.renderGraphPortrait	Passed	
StatisticsView.renderGraphLandscape	Passed	
StatisticsView.setLinks	Passed	
StatisticsView.setLinksLandscape	Passed	
StatisticsView.buildMenuLandscape	Passed	
StatisticsView.buildMenuPortrait	Passed	
StatisticsController.okPercentagem	Passed	
StatisticsController.stoppedPercentagem	Passed	
StatisticsController.notRunningPercentagem	Passed	
StatisticsController.unknownPercentagem	Passed	
StatisticsController.handleData	Passed	
StatisticsController.refreshGraph	Passed	
StatisticsController.handleRequest	Passed	
PlayerListView.renderView	Passed	
PlayerListController.handleRequest	Passed	
PlayerListController.getPlayers	Passed	
PlayerListController.handleData	Passed	
PlayerListController.previousPage	Passed	
PlayerListController.nextPage	Passed	
PlayerListController.togglePlayerSort	Passed	
PlayerView.renderView	Passed	
PlayerView.updateChannels	Passed	
PlayerController.handleRequest	Passed	
PlayerController.handleChannelData	Passed	
LogsListView.renderView	Passed	
LogsListView.renderLog	Passed	
LogsListView.closeLogPopUp	Passed	
LogsListController.getLogs	Passed	
LogsListController.handleRequest	Passed	
LogsListController.handleData	Passed	
LogsListController.previousPage	Passed	

Tabela 6.2: *Resultados dos testes de validação*

Classe.Função	Resultado	Erro
LogsListController.nextPage	Passed	
LogsListController.toggleSort	Passed	
WebServices.getPlayersWS	Passed	
WebServices.getLogsWS	Passed	
WebServices.getNetworkChannelsWS	Passed	
WebServices.setPlayerChannelWS	Passed	
RegExp.getPlayerId	Passed	
RegExp.getPlayerName	Passed	
RegExp.getNetworkId	Passed	
RegExp.getNetworkName	Passed	
RegExp.getRunning	Passed	
RegExp.getRunningSimplified	Passed	
RegExp.getCurrentSchedule	Passed	
RegExp.getUpdated	Passed	
RegExp.getUpdatedSimplified	Passed	
RegExp.getQueryString	Passed	
RegExp.getQueryStringV2	Passed	

Tabela 6.3: *Resultados dos testes de validação (Continuação)*

sível ter uma cobertura a 100%. Ao usar-se a *framework* jQtouch e outras bibliotecas JavaScript, como o jQuery, uma parte da aplicação fica necessariamente fora do alcance dos testes, mas isso acaba por ser negligenciável pois a *framework* e bibliotecas usadas já foram submetidas a testes rigorosos ao serem desenvolvidas. Portanto, o código abrangido pelos testes foi somente aquele que foi desenvolvido para esta aplicação em particular.

Na tabela 6.1 está apresentada a cobertura do código em termos das funções que foram submetidas aos testes. A primeira coluna contém o nome das classes, a segunda coluna contém o número de funções total de cada uma das classes correspondentes, a terceira coluna contém o número de funções que foram submetidas aos testes e a última coluna contém a percentagem de funções cobertas. A última linha da tabela é reservada para fazer o somatório de cada coluna. Podemos ver que houve uma cobertura a 100% das classes implementadas. Importa referir que a abrangência dos testes variou consoante a importância da função em concreto.

6.2 Testes de carga

Estavam previstos serem feitos testes de carga para simular a aplicação num uso intensivo, de modo a ver como é que ela se comporta e quais são os seus limites. Estes testes acabaram por não ser feitos devido a não ter controlo sobre a base de dados. Mas mesmo assim podem-se tirar previsões realistas do comportamento da aplicação, tendo em conta a arquitectura da mesma.

Em primeiro lugar é importante contextualizar o perfil de utilização desta aplicação. É uma aplicação de gestão e administração, o que faz com que seja usada por um grupo relativamente pequeno de pessoas. Portanto, relativamente ao número de utilizadores em simultâneo, pode-se prever que não vai ser suficientemente grande de modo a comprometer a usabilidade geral da aplicação no lado do servidor. Relativamente à arquitetura da aplicação, esta é bastante eficiente no que diz respeito aos pedidos ao servidor e à quantidade de informação (bytes) transmitida entre o servidor e o cliente. Esta eficiência é conseguida devido às novas funcionalidades do HTML5, nomeadamente a possiblidade de fazer *caching* da aplicação e outros dados no cliente. Isto faz com que só numa primeira instância ou se a aplicação tiver sido atualizada é que o servidor envia todo o código e dados da aplicação para o cliente, reduzindo assim drasticamente a carga no servidor. Todas as páginas ou *views* da aplicação são geradas no cliente fazendo com que os únicos pedidos feitos ao servidor sejam relativos à consulta da base de dados. E mesmo assim, a informação passada é minimizada uma vez que se utilizam paginações na listagem dos dados, além de que estas informações são armazenadas localmente de uma forma temporária para evitar vários pedidos da mesma informação sem esta ter mudado.

Portanto, tendo em conta a arquitetura e o modo de funcionamento da aplicação, a carga no lado do servidor em termos computacionais vai incidir quase exclusivamente no sistema gestor de bases de dados. Uma vez que para além de não ter acesso direto à base de dados, também não há qualquer informação relativa à estrutura e dimensão da mesma, é inviável criar uma base de dados local para testes de carga.

6.3 Testes de compatibilidade e usabilidade

Os testes de compatibilidade e usabilidade consistiram em testar a aplicação em diversos *smartphones* e *browsers*, com o objetivo de assegurar o seu correto funcionamento e visualização. A utilização de vários *browsers* prende-se com o facto de que nem todos eles suportam o mesmo conjunto de funcionalidades e, portanto, é necessário saber qual o impacto que isso pode provocar no funcionamento da aplicação. Para além disto, a aplicação foi testada em ecrãs com as dimensões mais frequentes em *smartphones*. Aqui, os emuladores tiveram um papel bastante importante ao permitirem criar dispositivos virtuais com as características que quiséssemos, nomeadamente a dimensão e resolução do ecrã. Os testes foram realizados manualmente e era testada a rapidez/fluidez da aplicação em geral, com ênfase para a interface. São testes com algum caráter subjetivo ao contrário dos testes de validação, anteriormente descritos.

Foram utilizados os seguintes *smartphones* nos testes realizados:

- Ipod touch
- Samsung galaxy S
- LG P500
- HTC Touch HD

Relativamente aos testes realizados nos *smartphones* e emuladores, estes serviram em primeira instância para validar a correta visualização da aplicação face às diferentes resoluções do ecrã. No geral a aplicação foi testada nas seguintes resoluções: 320x480, 480x640, 480x800, 640x960, 768x1024, 768x1280. Os resultados foram bastante positivos, uma vez que a aplicação se adaptou bem a todas essas resoluções. Verificou-se também que a resolução mínima recomendada é a de 320x480, sendo que a maior parte dos *smartphones* tem resoluções iguais ou superiores.

O passo seguinte foi testar a aplicação em diferentes *browsers*, para se poder aferir de que maneira a aplicação era afetada pela diferença de funcionalidades suportadas pelos mesmos. Ao todo, foram 6 os *browsers* utilizados, dando uma cobertura total das plataformas (iOS, Android e Windows Phone 7) em que a aplicação tem que ser suportada. Quanto aos parâmetros utilizados para classificar a aplicação, foram os seguintes:

- O correto *layout* das páginas.
- O suporte total dos estilos CSS3 utilizados.
- O suporte das *media queries* CSS3 utilizadas.
- O suporte das animações utilizadas na transição entre páginas.
- O suporte do elemento *Canvas* do HTML5.
- Fluídez da interface, do ponto de vista da percepção do utilizador.
- O correto funcionamento da lógica de negócio.

De entre estes parâmetros, destacam-se três que eram críticos e portanto tinham que passar nos testes, sendo eles: o correto *layout* das páginas, o correto funcionamento da lógica de negócio e o suporte do elemento *Canvas*. Nas tabelas 6.4 e 6.5 estão os resultados dos testes, que de seguida irão ser explicados. Quanto ao *layout* das páginas, como já era previsível, todos os *browsers* apresentaram-no de acordo com o esperado e portanto levaram todos nota positiva. Quanto ao suporte total dos estilos CSS3 utilizados, nem todos os *browsers* passaram no teste. Neste aspeto, o impacto não é muito grande uma vez que a única propriedade de estilo que não passou nos testes foi o *border-radius*, em que em vez de aparecerem os cantos arredondados em alguns elementos, apareciam cantos retos.

No suporte das *media queries* CSS3 utilizadas, todos os *browsers* tiveram nota positiva. A nível de *media queries* usaram-se apenas para atribuir estilos dinamicamente caso o aparelho estivesse no modo *landscape* ou *portrait*, o que significa que a aplicação nesses *browsers* vai disponibilizar as páginas de acordo com o esperado.

No suporte das animações utilizadas na transição entre páginas, três *browsers* não passaram no teste, no entanto esta funcionalidade está implementada mais pela estética e para ser visualmente apelativo, não sendo portanto crítica. A justificação destes resultados prende-se com o facto das animações serem feitas nativamente pelo motor *webkit*, e logicamente os *browsers* que não o usam ou que usam uma versão mais antiga desse motor não conseguem reproduzir as animações.

No que diz respeito ao suporte do elemento *Canvas* do HTML5, aspeto crítico, todos os *browsers* o suportam e portanto passaram no teste. Este aspeto era crítico porque o gráfico apresentado na página inicial é feito recorrendo ao elemento *Canvas*, e portanto, o não suporte desse elemento teria um impacto bastante negativo na aplicação.

Quanto à fluídez da interface, embora este seja um aspeto subjetivo, todos os *browsers* demonstraram uma boa experiência de utilização. A lógica de negócio, como já era de esperar, uma vez que é praticamete toda implementada em JavaSript, apresentou um correto funcionamento em todos os

	Android default browser	Opera mobile (12.0.1)	Dolphin (4.0)
Correto Layout das páginas	+	+	+
Suporte total dos estilos CSS3 utilizados	+	-	+
Suporte das *media queries* CSS3 utilizadas	+	+	+
Suporte das animações utilizadas	+	-	+
Suporte do elemento canvas do HTML5	+	+	+
Fluídez da interface	+	+	+
Correto funcionamento da lógica de negócio	+	+	+

Tabela 6.4: *Resultados dos testes efetuados nos browsers*

browsers. De realçar que os que não passaram em algum dos parâmetros, podem vir a suportá-los em versões futuras.

	Opera Mini (7.0.1)	Internet Explorer 9 mobile	Safari (4.0.5)
Correto Layout das páginas	+	+	+
Suporte total dos estilos CSS3 utilizados	-	-	+
Suporte das *media queries* CSS3 utilizadas	+	+	+
Suporte das animações utilizadas	-	-	+
Suporte do elemento canvas do HTML5	+	+	+
Fluídez da interface	+	+	+
Correto funcionamento da lógica de negócio	+	+	+

Tabela 6.5: *Resultados dos testes efetuados nos browsers (Contiuação)*

Capítulo 7

Conclusões

Do desenvolvimento do projeto conducente a esta tese e do estudo realizado na área do desenvolvimento de aplicações para *smartphones* emergem várias conclusões, sendo algumas delas de caráter subjectivo e, portanto, passíveis de interpretações diferentes.

Esta área é relativamente recente e o seu futuro aparenta ser bastante promissor. No entanto, devido ao ritmo acelerado da sua evolução, é uma área que ainda não tem as suas bases totalmente consolidadas e bem definidas. Existem, por exemplo, muitas *frameworks* para o desenvolvimento de aplicações web, mas nenhuma delas pode ser considerada verdadeiramente dinâmica, estável, sem bugs e com suficientes funcionalidades. É portanto um aspeto que deveria ser alvo de um esforço mais concentrado, pelo menos no sentido de se desenvolver uma *framework* base que fosse estável e dinâmica, de modo a ser um ponto de partida para futuras implementações ou extensões de funcionalidades. Um fator que está de certa forma a impôr limites no avanço desta área, é o facto do HTML5 ainda estar um pouco longe de ser totalmente suportado pelos vários *browsers*. Além disso, existe muita discrepância na quantidade e tipo de funcionalidades já suportadas pelos diversos *browsers* dos *smartphones*. Por exemplo, o que vem por defeito no android não suporta certas funcionalidades do HTML5, o que limita e condiciona o desenvolvimento de aplicações para esse *browser*. Tal não acontece tão marcadamente no iOS, onde o *Safari*, é um dos que mais funcionalidades suporta.

É evidente do estudo feito e da realização do projeto que existem muitos novos desafios quando se está a desenvolver uma aplicação web para *smartphones*. Desde o tamanho dos ecrãs até à realização de testes, em paralelo com a rápida evolução desta área, estes são exemplos de aspetos novos que não se podem desvalorizar sob pena de comprometer a aplicação desenvolvida. Neste sentido, foram analisadas várias *frameworks* que agilizam o desenvolvimento de uma parte fundamental da aplicação que é a interface, bem como referidas algumas linhas orientadoras para a modelar, focadas

principalmente no tipo de conteúdo e na sua organização.

Quanto ao projeto desenvolvido foram atingidos todos os objetivos delineados. Era pretendida uma aplicação web adaptada a *smartphones* para gerir e controlar uma rede de painéis digitais (*digital signage*). A aplicação foi concluída com sucesso, cumprindo com todos os requisitos propostos pela empresa *Ubisign*. No decorrer do desenvolvimento do projeto, e na perspetiva de quem já desenvolveu pequenas aplicações nativas para android, foi realmente notório o esforço adicional necessário para desenvolver uma aplicação web para *smartphones* em comparação com uma aplicação nativa. Este esforço adicional recaiu essencialmente no teste da aplicação e na garantia de esta funcionar corretamente nos diversos *browsers* atuais, que são bastante díspares quanto às funcionalidades suportadas. No entanto, maior esforço seria necessário para implementar a aplicação nativa para os vários sistemas operativos (Android, iOS, Windows Phone, etc). Tendo em conta o género de aplicação requerida, foi portanto uma boa decisão tê-la desenvolvido recorrendo a tecnologias web, que se revelaram perfeitamente adequadas aos objetivos e requisitos definidos.

Sem dúvida alguma que os próximos anos irão ter um grande impacto na forma como as pessoas interagem com os seus *smartphones*. Há ainda muito para aprender e também muitas mentalidades a adaptar a este novo conceito de aplicações. Apesar de todos os desafios e desvantagens inerentes, as aplicações web móveis estão aqui para ficar e a tendência é para se tornarem mais populares. Embora o futuro desta área seja aparentemente promissor, é ainda relativamente cedo para se poder fazer previsões realistas no médio e longo prazo. É certo que as bases estão lançadas e tem tudo para dar certo, mas vai também depender da sua adopção por parte dos utilizadores e principalmente pelas comunidades de desenvolvimento.

Referências Bibilográficas

[BMM+97] J. Bosch, P. Molin, M. Mattsson, P.O. Bengtsson, e M. Fayad. Object-oriented frameworks-problems & experiences. *Building Application Frameworks: Object Oriented Foundations of Framework Design*, 1:1–2, 1997.

[Bre02] Stephen Brewster. Overcoming the lack of screen space on mobile computers. *Personal and Ubiquitous Computing*, 6:188–205, 2002. 10.1007/s007790200019.

[CL11] Andre Charland e Brain Leroux. Mobile Application Development: Web vs. Native. *COMMUNICATIONS OF THE ACM*, 54(5):49–53, MAY 2011.

[Cle09] Eric K. Clemons. Business models for monetizing internet applications and web sites: Experience, theory, and predictions. *Journal of Management Information Systems*, 26(2):15 – 41, 2009.

[DV11] Subhankar Dhar e Upkar Varshney. Challenges and business models for mobile location-based services and advertising. *Commun. ACM*, 54(5):121–128, Maio 2011.

[FF11] S. Fulton e J. Fulton. *HTML5 Canvas*. O'Reilly Series. O'Reilly Media, 2011.

[FLWZ07] Georey C. Fox, Xiaoming Li, Yuhong Wen, e Guansong Zhang. Studies of integration and optimization of interpreted and compiled languages. 2007.

[FS97] Mohamed Fayad e Douglas C. Schmidt. Object-oriented application frameworks. *Commun. ACM*, 40:32–38, October 1997.

[GBD00] W D Gray e D A Boehm-Davis. Milliseconds matter: an introduction to microstrategies and to their use in describing and predicting interactive behavior. *Journal of experimental psychology Applied*, 6(4):322–335, 2000.

[GHJV95] Erich Gamma, Richard Helm, Ralph E. Johnson, e John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[GT04] Jun Gong e Peter Tarasewich. Guidelines for handheld mobile device interface design. Em *In Proceedings of the 2004 DSI Annual Meeting*, 2004.

[HF98] David S. Hamu e Mohamed E. Fayad. Achieving bottom-line improvements with enterprise frameworks. *Commun. ACM*, 41(8):110–113, Agosto 1998.

[HO11] Adrian Holzer e Jan Ondrus. Mobile application market: A developer's perspective. *Telematics and Informatics*, 28(1):22 – 31, 2011.

[Hol11] A.T. Holdener. *HTML5 Geolocation*. Oreilly and Associate Series. O'Reilly Media, 2011.

[Hoy11] Matthew B. Hoy. Html5: A new standard for the web. *Medical Reference Services Quarterly*, 30(1):50–55, 2011.

[Joh97] Ralph E. Johnson. Frameworks = (components + patterns). *Commun. ACM*, 40:39–42, October 1997.

[Lar04] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[LS11] B. Lawson e R. Sharp. *Introducing Html5*. New Riders, 2011.

[MBF99] Michael Mattsson, Jan Bosch, e Mohamed E. Fayad. Framework integration problems, causes, solutions. *Commun. ACM*, 42(10):80–87, Outubro 1999.

[Mul11] C. Mulder. Challenges of web application development: How to optimize client-side code. 2011.

[VN10] S.J. Vaughan-Nichols. Will html 5 restandardize the web? *Computer*, 43(4):13 –15, april 2010.

Referências Web

[And10] Rachel Andrew. How to use css3 media queries to create a mobile version of your website. http://www.smashingmagazine.com/2010/07/19/how-to-use-css3-media-queries-to-create-a-mobile-version-of-your-website/, 2010.

[Cor11] Ubisign Corporation. About us. http://http://www.ubisign.com/public2/PortalRender.aspx?PageID=6c9fa15e-6938-4492-85cf-df335eb37148, 2011.

[Gil12] Sean Gilligan. iui - web ui framework for mobile devices. http://code.google.com/p/iui/, 2012.

[Hei10] Chris Heinen. iui vs. jqtouch. http://heinencreative.com/articles/iui-vs-jqtouch, 2010.

[Hew07] Joe Hewitt. Introducing iui. http://joehewitt.com/2007/07/11/introducing-iui, 2007.

[Kan10] David Kaneda. jqtouch - jquery plugin for mobile web development. http://jqtouch.com/, 2010.

[Kan12a] David Kaneda. Animations - jqtouch - how to use built-in animations. http://code.google.com/p/jqtouch/wiki/Animations, 2012.

[Kan12b] David Kaneda. Callback events - jqtouch - built-in triggers/events in jqtouch including page animations and touch events. http://code.google.com/p/jqtouch/wiki/CallbackEvents, 2012.

[Kan12c] David Kaneda. Offline support - jqtouch - quick tips on making jqtouch offline-capable. http://code.google.com/p/jqtouch/wiki/OfflineSupport, 2012.

[Pho11] PhoneGap. How phonegap plays an important part in our enterprise offering. http://phonegap.com/2011/06/27/how-phonegap-plays-an-important-part-in-our-enterprise-offering/, 2011.

[Pho12a] PhoneGap. About phonegap. http://phonegap.com/about, 2012.

[Pho12b] PhoneGap. Phonegap build. https://build.phonegap.com/, 2012.

[Pho12c] PhoneGap. Supported features - phonegap. http://phonegap.com/about/features, 2012.

[Raa10] Jon Raasch. How to build a mobile website. http://www.smashingmagazine.com/2010/11/03/how-to-build-a-mobile-website/, 2010.

[Sne09] Steven Snell. Mobile web design trends for 2009. http://www.smashingmagazine.com/2009/01/13/mobile-web-design-trends-2009/, 2009.

[W3C12a] W3C. Html5 differences from html4. http://www.w3.org/TR/2012/WD-html5-diff-20120329/, 2012.

[W3C12b] W3C. Html5 geolocation api specification. http://dev.w3.org/geo/api/spec-source.html, 2012.

[Wil12] Hamish Willee. Mobile design pattern: Accordion menu. http://www.developer.nokia.com/Community/Wiki/Mobile_Design_Pattern:_Accordion_Menu, 2012.

Anexos

Super Use Case			
Author	Joaquim Anacleto		
Date			
Brief Description	Autenticação no sistema		
Preconditions	Utilizador não está autenticado no sistema.		
Post-conditions	**Sucesso:** Autenticação efectuada com sucesso **Insucesso:** Autenticação não é efectuada.		
Flow of Events		**Actor Input**	**System Response**
	1	Utilizador acede ao site	
	2		Sistema redireciona o utilizador para o formulário de login
	3	Utilizador introduz username e password	
	4		Sistema autentica utilizador no sistema
	5		Utilizador é redirecionado para a Homepage
Alternativa 1		**Actor Input**	**System Response**
	1		
	2		
	3		
	4		Sistema devolve mensagem de insucesso na autenticação devido a username/password incorrectos.
	5		Volta para 3

Figura 1: *Especificação textual de Use Cases: Login*

Super Use Case			
Author	Joaquim Anacleto		
Date			
Brief Description	Listagem dos players.		
Preconditions	Utilizador autenticado no sistema.		
Post-conditions			
Flow of Events		**Actor Input**	**System Response**
	1	Utilizador acede ao site	
	2		Sistema retorna a homepage, que contém as estatísticas do estado geral da rede de players.
	3	Utilizador clica no link onde está explícito o número de players total.	
	4		Sistema retorna a lista geral de players sem nenhuma filtragem.
Alternativa 1: Ver players OK		**Actor Input**	**System Response**
	1	Utilizador acede ao site	
	2		Sistema retorna a homepage, que contém as estatísticas do estado geral da rede de players.
	3	Utilizador clica no link onde está explícito o número de players que têm como status "OK".	
	4		Sistema retorna a lista de players que tenham como status "OK".
Alternativa 2: Ver players Stopped		**Actor Input**	**System Response**
	1	Utilizador acede ao site	
	2		Sistema retorna a homepage, que contém as estatísticas do estado geral da rede de players.
	3	Utilizador clica no link onde está explícito o número de players que têm como status "Stopped".	
	4		Sistema retorna a lista de players que tenham como status "Stopped".
Alternativa 3: Ver players not running		**Actor Input**	**System Response**
	1	Utilizador acede ao site	
	2		Sistema retorna a homepage, que contém as estatísticas do estado geral da rede de players.
	3	Utilizador clica no link onde está explícito o número de players que têm como status "Not Running".	
	4		Sistema retorna a lista de players que tenham como status "Not Running".
Alternativa 4: Ver players unknown		**Actor Input**	**System Response**
	1	Utilizador acede ao site	
	2		Sistema retorna a homepage, que contém as estatísticas do estado geral da rede de players.
	3	Utilizador clica no link onde está explícito o número de players que têm como status "Unknown".	
	4		Sistema retorna a lista de players que tenham como status "Unknown".

Figura 2: *Especificação textual de Use Cases: Listagem dos players*

Super Use Case			
Author	Joaquim Anacleto		
Date			
Brief Description	Alteração do canal de um player.		
Preconditions	Utilizador autenticado no sistema. Utilizador está a visualizar os detalhes de um determinado player.		
Post-conditions	**Sucesso:** Canal sintonizado pelo player é alterado. **Insucesso:** Canal sintonizado pelo player não é alterado		
Flow of Events		**Actor Input**	**System Response**
	1	Utilizador acede à página que disponibiliza os detalhes de um determinado player.	
	2		Sistema devolve a página de detalhes do player selecionado e disponibiliza sob a forma de uma select list os canais disponiveis para aquele player.
	3	Utilizador altera o canal sintonizado pelo player selecionando outro canal disponível na Select list (drop-down list).	
	4		Sistema altera o canal sintonizado por esse player para o canal escolhido pelo utilizador
Alternativa 1: Erro ao tentar mudar de canal		**Actor Input**	**System Response**
	1		
	2		
	3		
	4		Sistema retorna mensagem de erro por não conseguir alterar o canal sintonizado pelo player.

Figura 3: *Especificação textual de Use Cases: Alterar canal Use Case.*

Super Use Case			
Author	Joaquim Anacleto		
Date			
Brief Description	Ver detalhes de um player		
Preconditions	Utilizador autenticado no sistema.		
Post-conditions	Display dos detalhes do player selecionado		
Flow of Events		**Actor Input**	**System Response**
	1	Utilizador acede à listagem dos players.	
	2	Utilizador seleciona (clicando) o player sobre o qual quer vizualizar os seus detalhes.	
	3		Sistema devolve a página que lista todos os detalhes do player selecionado.
Alternativa		**Actor Input**	**System Response**
	1	Utilizador faz uma procura por um determinado player.	
	2		Sistema devolve a lista de players que resultaram da pesquisa
	3	Utilizador seleciona (clicando) o player sobre o qual quer vizualizar os seus detalhes.	
	4		Sistema devolve a página que lista todos os detalhes do player selecionado.

Figura 4: *Especificação textual de Use Cases: Ver detalhes de um player.*

Super Use Case			
Author	Joaquim Anacleto		
Date			
Brief Description	Consultar logs de um player		
Preconditions	Utilizador autenticado no sistema		
Post-conditions	Visualização dos logs do player selecionado		
Flow of Events		**Actor Input**	**System Response**
	1	Utilizador seleciona o player, do qual quer visualizar os logs.	
	2		Sistema devolve a página dos detalhes do player onde tem um link para visualizar os logs.
	3	Utilizador clica no link dos logs	
	4		Sistema devolve a página com o registo dos log associado ao player em questão.

Figura 5: *Especificação textual de Use Cases: Consultar logs de um player*

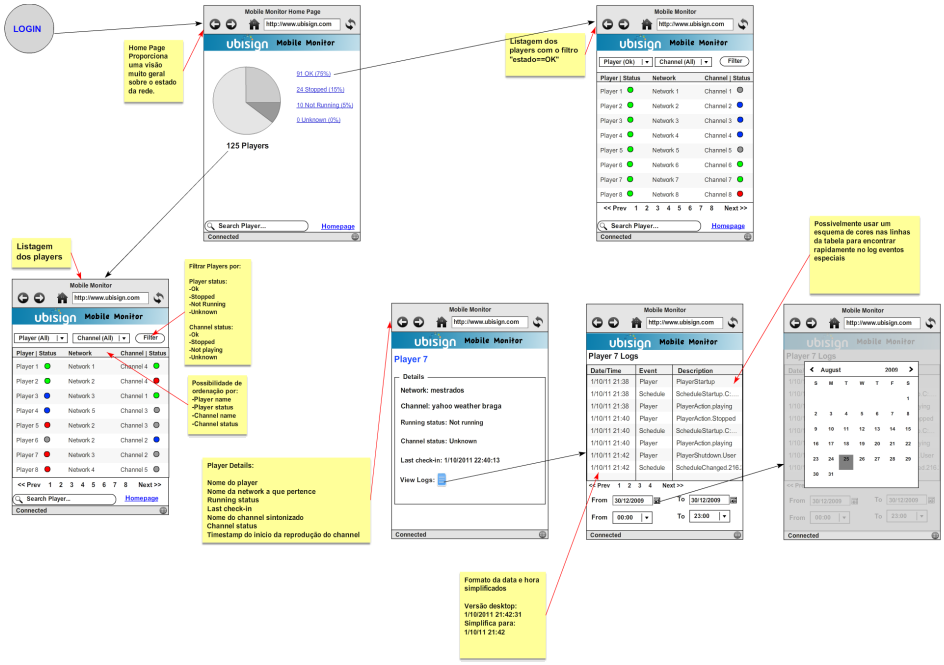


Figura 6: *Mockups da interface*