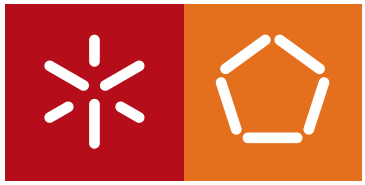
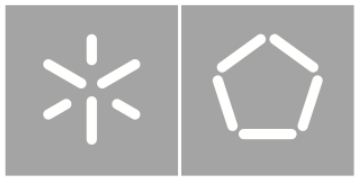


Universidade do Minho
Escola de Engenharia

Ana Isabel Sampaio

Responsive Web Design

Outubro de 2013



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ana Isabel Sampaio

Responsive Web Design

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Professor José Francisco Creissac Campos

Outubro de 2013

Este trabalho foi financiado por Fundos FEDER através do Programa Operacional Fatores de Competitividade – COMPETE e por Fundos Nacionais através da FCT – Fundação para a Ciência e a Tecnologia no âmbito do projeto FCOMP-01-0124-FEDER-015095.

Abstract

From computers to tablets and smartphones, the term ubiquity, combined with the diversity of devices available in the market, has changed the way we access and share information. It has become more and more important to offer solid user experiences to an increasing number of contexts.

Responsive Web Design offers a set of tools to support the creation of web pages that adapt to any screen size. We use fluid grids, flexible images and media queries to adapt web pages regardless of the device's screen dimensions.

Hoping to contribute with new resources to the responsive design area, this dissertation describes how a tool was developed which allows web sites' navigation to automatically adapt, according to different patterns. The main goal is to make designers' and developers' work easier by providing adaptive menus that adjust to the screen size.

However, different use contexts cannot be limited to the screen size. Thinking on the specific case of mobile devices, which are with us wherever we go, different connectivity conditions (from strong wireless networks to limited 3G connections) may have an impact on the user experience. That's why optimization is such an important topic in responsive web design. Furthermore, touch screens open new opportunities to interact directly with the website and its content. Taking advantage of this new set of interactions, the featured tool reflects this optimization to approach user experience to each context.

Resumo

Dos computadores aos *tablets* e *smartphones*, a noção de ubiquidade, aliada à diversidade de dispositivos disponíveis no mercado, mudou a forma como acedemos e partilhamos informação nos dias de hoje. É cada vez mais importante oferecer experiências de utilização consistente para um crescente número de contextos.

O *Responsive Web Design* oferece um conjunto de ferramentas que permitem criar páginas *web* que respondem ao tamanho de qualquer ecrã. São utilizadas *grids* fluidas, imagens flexíveis e *media queries* para adaptar as páginas *web* independentemente das dimensões de ecrã do dispositivo.

Na expetativa de contribuir com novos recursos para a área do *responsive design*, ao longo desta dissertação é descrito o desenvolvimento de uma ferramenta que permite adaptar automaticamente a navegação dos *sites* e *aplicações web*, tendo em conta diferentes soluções padrão. O seu objetivo consiste em facilitar o trabalho dos *designers* e *developers,* apresentando sempre um menu adaptado ao tamanho do ecrã.

No entanto, os diferentes contextos de utilização não se podem limitar apenas ao tamanho do ecrã. Pensando no caso concreto dos dispositivos móveis, que nos acompanham para qualquer lugar, diferentes condições de conectividade (desde fortes redes *wireless*, a conexões 3G limitadas) podem ter impacto na experiência de utilização. A otimização é, por isso, um fator importante em *responsive web design*. Além disso, os ecrãs sensíveis ao toque abrem novas oportunidades para interagir com o *site* e com o seu o conteúdo. De forma a tirar partido deste tipo de interações, a ferramenta desenvolvida reflete esta otimização, tentando assim aproximar a experiência de utilização ao contexto utilizado.

Agradecimentos

Gostaria de agradecer em primeiro lugar ao professor José Creissac Campos, pela orientação durante todos estes meses, ajudando sempre a definir qual o melhor caminho a seguir. O conhecimento científico, o seu trabalho e dedicação foram essenciais para a concretização desta dissertação de mestrado.

Agradeço ainda à professora Teresa Galvão, por se ter disponibilizado a ver o meu trabalho e pelas sugestões dadas.

Gostaria de agradecer também à Engenheira Beatriz Oliveira da Bind, que me ajudou na escolha do tema. O seu trabalho nesta área converteu-se numa motivação para esta dissertação. Agradeço também aos meus colegas da Bind, João e Pedro, pelo espirito de companheirismo.

Por fim, agradeço aos meus pais, que sempre me apoiaram durante esta jornada. À minha mãe pela paciência e compreensão, e ao meu pai pela perseverança e preocupação demonstradas até ao final.

Conteúdo

Lista de figuras

Lista de tabelas

Abreviaturas

3G	3rd Generation mobile telecommunications
API	Application Programming Interface
AUI	Adaptive User Interfaces
AWD	Adaptive Web Design
CSS	Cascading Style Sheets
HCI	Human-Computer Interaction
HTML	HyperText Markup Language
DDR	Device Description Repository
DOM	Document Object Model
RWD	Responsive Web Design
Sass	Syntactically Awesome Stylesheets
UI	User Interface
UX	User Experience
W3C	World Wide Web Consortium
WAI-ARIA	Web Accessibility Initiative - Accessible Rich Internet Applications

Capítulo 1

Introdução

1.1 Contextualização

Com a chegada da *World Wide Web*[1], surge um novo conceito de acesso à informação, e o uso de *websites* passou a tornar-se bastante comum. Desde essa altura que a criação de aplicações *web* tem vindo a disparar, e cada vez mais surgem tecnologias que facilitam o processo de criação e manutenção destas aplicações. De acordo com Royal Pingdom (2013), em 2010 foram lançados 21.4 milhões novos *sites*, no ano seguinte cerca de 28 milhões e em 2012 foram 51 milhões os novos *sites* criados.

O modo de acesso à *web* tem vindo a evoluir, e se inicialmente se fazia apenas a partir de computadores *desktop*, assiste-se agora a um aumento cada vez mais significativo do acesso a partir de dispositivos móveis. É esperado que a navegação através de dispositivos móveis ultrapasse o acesso à *web* a partir de computadores nos próximos anos. No entanto, a diversidade de plataformas coloca desafios ao desenvolvimento de aplicações *web*, pois é necessário que estas se adaptem às características dos diferentes dispositivos (de modo particular, ao tamanho dos seus ecrãs). É neste contexto que surge o conceito de *Responsive Web Design* (Marcotte 2010).

Responsive é uma técnica que permite a navegação de *websites* a partir de uma variedade de dispositivos, desde *smartphones* e *tablets* a computadores *desktop*, ajustando a sua estrutura ao tamanho da janela de visualização, de forma a proporcionar uma boa experiência de utilização no maior número de dispositivos possível.

No livro *Responsive Web Design*, de Ethan Marcotte (2011), o autor antecipa e dá resposta a esta necessidade de adaptação a qualquer dispositivo. Ao longo da sua obra, Marcotte explora técnicas de CSS (*Cascading Style Sheets*) e princípios de *design*, incluindo *grids* fluidas, imagens flexíveis e *media queries*, demonstrando ser possível dar uma experiência de utilização de qualidade aos utilizadores, independentemente do tamanho do ecrã.

[1] Ao longo desta dissertação são utilizados frequentemente alguns termos em Inglês porque não foi encontrada uma tradução adequada em Português.

1.2 Motivação

A *web* gira em torno de informação. A cada dia, em cada *site*, a informação é divulgada, partilhada, solicitada e recolhida. A troca de informação tem sido crucial para o crescimento da *web*, e certamente que esta expansão tem afetado o trabalho e a vida das pessoas que permanecem conectadas. Atualmente, de acordo com Internet World Stats (2012), são mais de 2,4 mil milhões de pessoas em todo o mundo. A constante partilha de informação tem contribuído para que as pessoas anseiem estar permanentemente ligadas, em qualquer hora e em qualquer local. A chegada dos *smarphones* e *tablets* potencializou este cenário. De acordo com um estudo recente publicado pela Yahoo!, 59% dos utilizadores visitam um *site* no seu telemóvel e depois fazem *follow up* mais tarde no seu computador. Por outro lado, 34% fazem o oposto (Yahoo! 2011). Uma questão que se levanta é a de as aplicações estarem ou não realmente preparadas para oferecerem uma experiência de utilização semelhante quando visualizadas em qualquer tipo de dispositivos.

A capacidade de adaptação das aplicações é já um requisito imposto pela maior parte dos utilizadores. As empresas assumem que é fundamental para o negócio disponibilizar o seu *site* em dispositivos móveis. Os utilizadores esperam poder facilmente aceder a um *site* a partir do seu *smartphone*, com uma experiência de utilização tão intuitiva como no *desktop*. Garantir o acesso à informação em qualquer altura, através de qualquer plataforma, tornaram-se requisitos cada vez mais comuns. Uma solução que permite criar aplicações adaptativas conforme o contexto do utilizador é o *responsive web design*.

Uma alternativa ao *responsive web design* é construir aplicações específicas para cada plataforma móvel. No entanto, construir aplicações específicas para as diferentes plataformas não parece ser uma opção razoável para pequenas empresas, em que o desenvolvimento de uma aplicação móvel se torna um grande desafio, e o investimento necessário é geralmente muito elevado. Acresce que, por vezes, os utilizadores têm preferência em aceder à informação através do *browser*, como relata o estudo feito pelo (Pew Research Center 2012), em que 60% dos utilizadores de *tablets* preferem ler as notícias a partir da *web* do que utilizando a aplicação nativa disponibilizada.

Um outro contexto em que o desenvolvimento de aplicações *web* é preferível ao desenvolvimento de aplicações nativas, é o da prototipagem rápida de interfaces. Este é o caso do projeto APEX (Silva, et al. 2010), em que este trabalho se enquadra. A APEX é uma

plataforma de prototipagem de ambientes de computação ubíqua recorrendo a ambientes virtuais e que utiliza aplicações *web* para suportar a prototipagem de aplicações e o seu teste, quer dentro de ambientes virtuais (utilizando o *browser* como dispositivo virtual), quer em diferentes dispositivos móveis. Existindo a necessidade de adaptar os protótipos ao maior número de plataformas possível, o *responsive web design* pode ser visto como uma solução a adotar no desenvolvimento das interfaces.

1.3 Objetivos

Pretende-se, ao longo deste trabalho, investigar os princípios em que se baseia o conceito de *responsive web design*, bem como identificar os problemas associados e o impacto que têm na experiência de utilização. Pretende-se também estudar diferentes padrões de adaptação e apurar a forma como se pode alcançar cada um deles.

Por fim, será implementada uma ferramenta que segue um conjunto de regras capazes de automatizar a adaptação de menus, tendo em contas os padrões de adaptação estudados. O seu objetivo principal consiste em agilizar esta tarefa, contribuindo assim para acelerar o processo de adaptação de *sites* ao tamanho do ecrã do dispositivo utilizado.

1.4 Estrutura da dissertação

No presente capítulo foi feito um enquadramento ao tema, seguido da motivação e dos objetivos que se pretendem atingir.

O capítulo seguinte contém a revisão da literatura relativamente às interfaces auto-adaptáveis, referindo a sua importância e alguns problemas associados. Posteriormente, o foco segue para as interfaces adaptativas no contexto da *web*, onde surge o conceito de *adaptive web design,* e mais tarde, *responsive web design.* São também apresentadas algumas *frameworks* de construção de interfaces baseadas em *responsive web design.*

No terceiro capítulo são abordados padrões de adaptação e sugestões de otimização conforme o contexto de utilização. No quarto capítulo é apresentada a ferramenta desenvolvida para adaptação automática da navegação de um *site*, e que integra todas as soluções padrão estudadas no capítulo anterior; no capítulo seguinte são apresentados exemplos de utilização da ferramenta. Por fim, o último capítulo é dedicado às conclusões e a sugestões de trabalho futuro.

Capítulo 2

Interfaces adaptativas e responsive web design

2.1 Interfaces adaptativas

Segundo (Rothrock, et al. 2002), interfaces adaptativas ou auto-adaptáveis (*Adaptive User Interfaces* – AUI) podem ser definidas como sistemas que adaptam o que é apresentado no ecrã e as ações disponíveis para o utilizador, aos objetivos e ao estado ou situação atual do sistema. A utilização de AUIs tem como pressuposto melhorar a interação do utilizador com o sistema, facilitando, através de uma adaptação automática às necessidades do primeiro, a utilização do segundo. O pressuposto é que interfaces adaptadas podem providenciar um melhor desempenho do utilizador na utilização dos sistemas, ao auxiliá-los, por exemplo, a superar dificuldades ou até mesmo problemas cognitivos em sistemas mais complexos (Lavie e Meyer 2010).

Existem vários estudos sobre a adaptação de interfaces, em âmbitos bastante distintos, como são os casos da customização de interfaces com menus adaptáveis em ambientes Windows (Gajos, et al. 2006), ou a avaliação do desempenho de tarefas com diferentes níveis de adaptação em contexto automóvel, por parte dos condutores (Lavie e Meyer 2010). Ainda que em contextos diferentes, estes estudos têm como objetivo comum melhorar a experiência de utilização dos sistemas onde se enquadram, para que o utilizador seja capaz de desempenhar as suas tarefas com menos esforço e mais eficácia. Dar ênfase ou facilitar o acesso a certas funcionalidades mais frequentes pode ser uma das adaptações.

2.1.1 Múltiplos contextos de utilização e plasticidade

Nos dias de hoje o acesso ubíquo à informação é uma necessidade crescente. Um dos requisitos que se impõe é a possibilidade de trabalhar em diferentes dispositivos, com características físicas diferentes, sem ser necessário redesenhar e reimplementar todo o

sistema, ao mesmo tempo que é assegurada a usabilidade da interface. Ao encontro desta necessidade surge o conceito de plasticidade de interfaces gráficas do utilizador, introduzido por (Thevenin e Coutaz 1999). O termo plasticidade foi inspirado nas propriedades dos materiais plásticos, que expandem e retraem sobre condições naturais sem quebrar, preservando assim as condições de utilização.

Em HCI (*Human-Computer Interaction*), interfaces plásticas são definidas como um tipo particular de interfaces adaptáveis, onde a especificação de uma única interface serve múltiplos contextos com variações físicas entre eles, como por exemplo diferentes tamanhos de ecrã. O objetivo deste tipo de interfaces é garantir a usabilidade sob as variações das características físicas do sistema, e ao mesmo tempo minimizar os custos de desenvolvimento e manutenção. Uma investigação mais profunda sobre plasticidade de interfaces pode ser consultada em (Thevenin e Coutaz 1999).

No seguimento da diversidade de contextos de utilização, surge a Cameleon Framework (ver Figura 1), uma *framework* de referência para interfaces sensíveis ao contexto. Ou seja, interfaces com capacidade para suportar múltiplos contextos de uso, reagindo e adaptando-se a alterações do mesmo. O contexto de uso pode ser decomposto em 3 facetas: o utilizador, a plataforma computacional (que inclui o tipo de dispositivos utilizados), e o ambiente em que o utilizador interage com as plataformas especificadas.

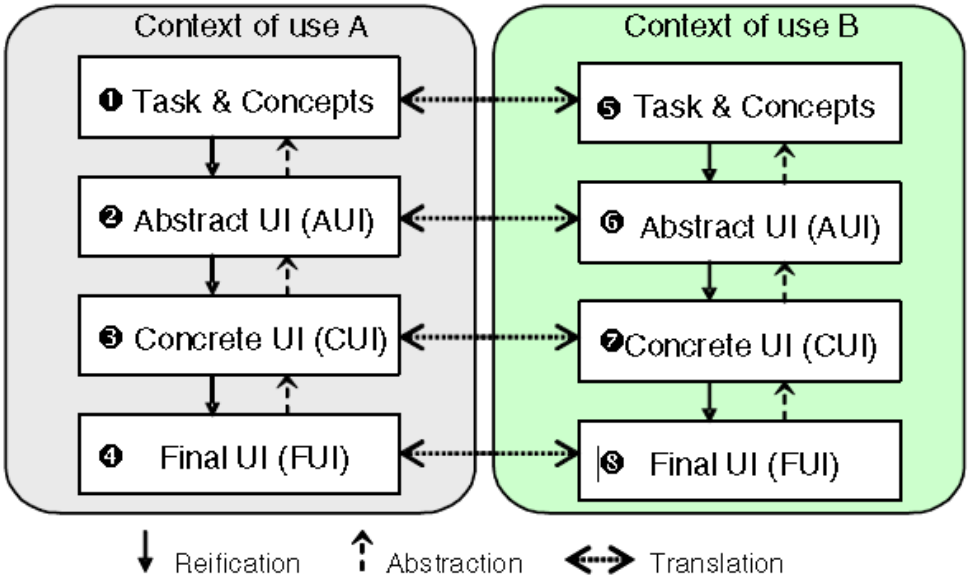


Figura 1. Cameleon Framework (Model-based User Interfaces Incubator Group 2010)

Para cada contexto identificado, esta *framework* estrutura o ciclo de vida do desenvolvimento de interfaces em quatro níveis de especificação:

- Modelo de tarefas e conceitos.
- Interface abstrata.
- Interface concreta.
- Interface final.

Perante a forma como se estruturam os níveis apresentados, podem ser estabelecidos três tipos de relações: reificação (transformação de um nível mais abstrato para um mais concreto), abstração (transformação de um nível mais concreto para um mais abstrato, utilizando engenharia reversa) e, por fim, translação (migração de um contexto de uso para outro).

2.1.2 Adaptação manual e automática de interfaces

A capacidade de uma interface se alterar dá origem a dois conceitos: adaptabilidade e adaptatividade. A adaptabilidade é a capacidade do sistema permitir ao utilizador customizar a interface, através de um conjunto de parâmetros pré-definidos. Por outro lado, adaptatividade é a capacidade da interface se adaptar automaticamente, isto é, sem a ação deliberada do utilizador (Thevenin e Coutaz 1999).

Estes dois conceitos podem ser complementares, como acontece na CrowdAdapt, uma ferramenta de *web design* sensível ao contexto que pretende dar suporte aos *developers* na criação de soluções de interfaces auto-adaptáveis, com base no conceito de *crowdsourcing*. Isto significa que os próprios utilizadores finais podem adaptar a interface ao seu contexto específico de utilização, de acordo com as suas preferências, de forma a explorar o espaço do ecrã disponível (Nebeling, Speicher e Norrie 2013). Um exemplo da sua utilização encontra-se na Figura 2.

.

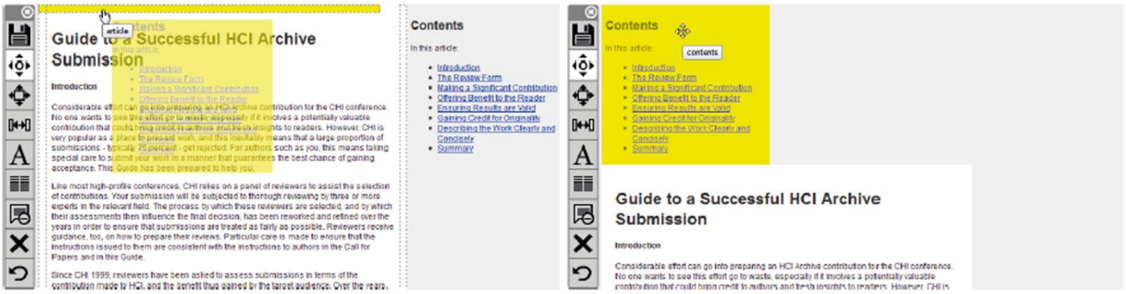


Figura 2. Utilização da ferramenta CrowdAdapt

A ferramenta CrowdAdapt tira proveito dos resultados da adaptação feita por parte dos utilizadores (adaptabilidade), reunindo informação para que posteriormente seja possível criar uma interface final com capacidades adaptativas. Tendo como referência a Cameleon Framework, a adaptação aos diferentes contextos é feita diretamente na interface final e, por isso, não é necessário recorrer a um modelo mais abstrato para que a adaptação ocorra.

Como vimos na Secção 2.1.1, o contexto de uso pode decompor-se em várias facetas – o utilizador, a plataforma e o ambiente de utilização. No caso concreto do contexto *web*, o tipo de plataforma ou dispositivo utilizado tem uma importância acrescida. Existe uma grande variedade de *browsers* com diferentes características entre si, e um crescente número de dispositivos com formas de interação distintas, o que dificulta a criação de interfaces *web* plásticas, isto é, interfaces que mantêm a usabilidade ao longo dos diferentes contextos em que são utilizadas. A secção seguinte trata com mais detalhe a adaptatividade das interfaces *web*.

2.2 Adaptive Web Design

Adaptive Design consiste na noção de que a experiência de utilização pode variar com base em fatores funcionais e ambientais, como é o caso das capacidades do *browser* ou propriedades do dispositivo. Por exemplo, a funcionalidade de localização geográfica permite que os dispositivos forneçam serviços adaptados à localização atual do utilizador. Com base no princípio de *adaptive design*, seria necessário verificar a existência desta funcionalidade e posteriormente adaptar em concordância. Por exemplo, introduzir uma opção adicional visível na interface (Oliveira 2013).

2.2.1 Framework para interfaces web adaptativas

A variação das características do *browser* e das propriedades do dispositivo devem produzir um impacto no aspeto da interface. Ao detetar essas variações, a interface deverá adaptar-se ao novo contexto do utilizador, nesse momento. Assim, mais que contextos de utilização claramente distintos, temos uma contínua variação do contexto e, para além de interfaces distintas para cada contexto, necessitamos de uma interface capaz de se adaptar a essa variação. De forma a suportar estes múltiplos contextos *web*, propõe-se uma nova aproximação à *Cameleon framework* (ver Figura 3).

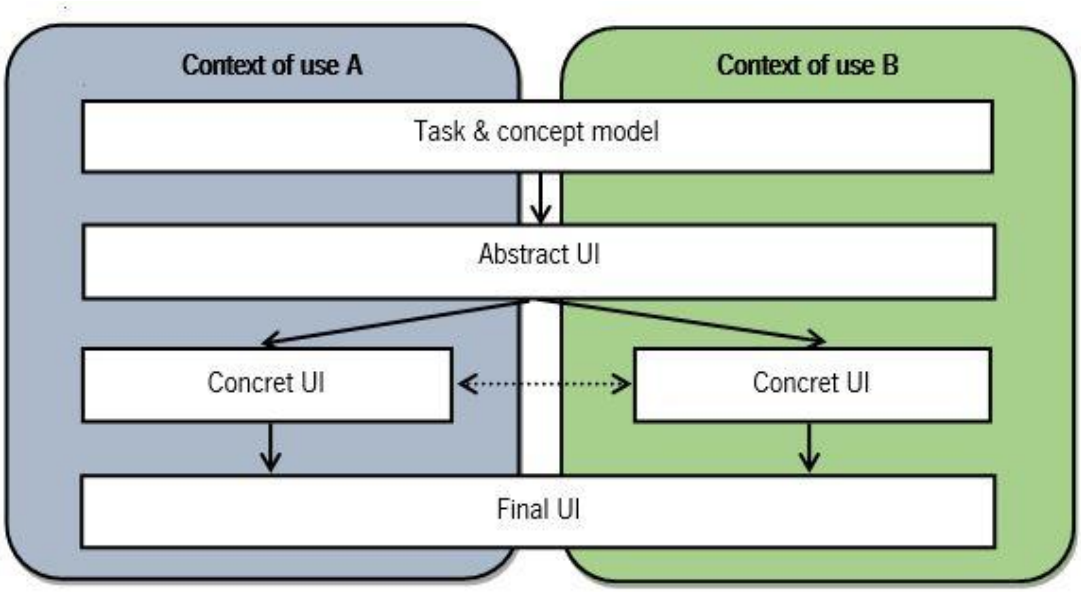


Figura 3. *Framework* para interfaces *web* adaptativas[2]

O contexto utilizado deverá ter impacto na forma como se apresenta visualmente a interface. Por exemplo, o tamanho das imagens e o tamanho da letra utilizados deverão adaptar-se ao tamanho do ecrã do dispositivo utilizado. Para além disso, certas funcionalidades podem ser adicionadas ou retiradas conforme as características disponíveis, como é o caso da localização geográfica. A forma de interação com a interface pode também variar, no caso de se utilizar o rato, o teclado ou o toque.

[2] Na figura estão apenas representadas as ligações de reificação (seta contínua) e translação (seta tracejada).

As variações do contexto devem refletir-se na interface que o utilizador vê. Deste modo, para diferentes contextos podemos desenhar interfaces distintas (*mockups*), e assim recorrer a diferentes interfaces concretas. A deteção do contexto atual é feita ao nível da interface final. A interface final, que diz respeito ao código da interface a correr no *browser*, deve ser inteligente para ser capaz de detetar diferentes características, e, em função disso, apresentar a interface adaptada correspondente. A Secção 4.1 explora formas de deteção das características do *browser* e do dispositivo.

Com a abordagem apresentada na Figura 3, o mesmo código que define a interface final é capaz de produzir interfaces que se adaptam automaticamente ao contexto de utilização. Essa capacidade pode resultar na criação de um nível de abstração adicional, que corresponde à interface visível para o utilizador num determinado contexto, ao qual pode ser designada *Visual User Interface* – ver Figura 4.

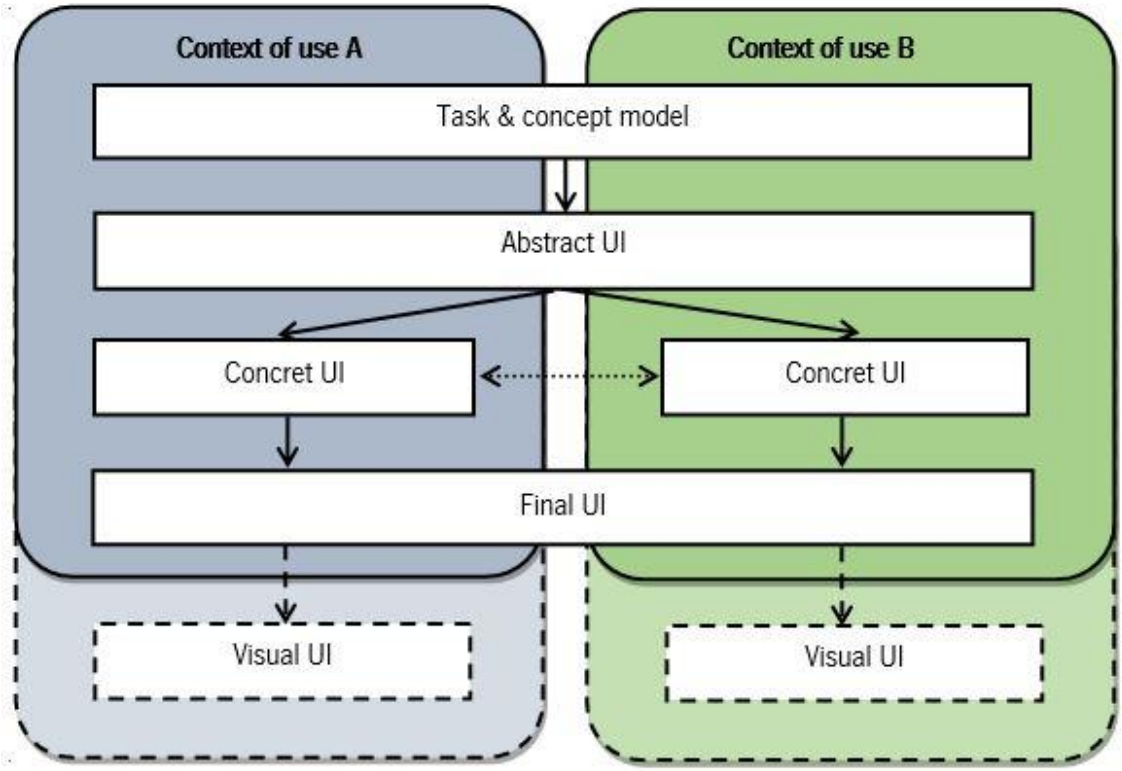


Figura 4. *Framework* para interfaces *web* adaptativas incluindo um nível de reificação adicional, Visual UI

O modelo apresentado corresponde à noção de interface *web* adaptativa. Note-se que uma alternativa seria manter uma interface final para cada interface concreta, colocando a lógica de

adaptação no servidor (que iria decidir qual a interface a enviar ao *browser* em função das suas características). No entanto isso implicaria manter várias interfaces finais e afastaria a lógica de adaptação ao dispositivo, do próprio dispositivo e das suas características.

Na secção seguinte é abordado o fundamento que permite que a adaptação seja feita ao nível da interface final, e não seja necessário definir diferentes interfaces finais para diferentes contextos *web*.

2.2.2 Progressive enhacement

O conceito de *Adaptive Web Design* tem como fundamento o termo *progressive enhacement*. *Progressive enhacement* é uma filosofia que visa criar níveis de experiência de utilização progressivamente mais ricos, mantendo o acesso ao conteúdo livre de restrições tecnológicas, isto é, independente do *browser* ou dispositivo utilizado, e também da versão de HTML ou CSS utilizada.

Progressive enhacement baseia-se no princípio fundamental da tolerância a falhas. A tolerância a falhas é a capacidade de um sistema continuar a operar quando ocorre um erro inesperado. As tecnologias *web*, nomeadamente as linguagens HTML e CSS, tiram benefício da tolerância a falhas, uma vez que foram especificadas de forma a serem ignoradas nas situações de incompreensão por parte do *browser*. A forma como estas linguagens foram especificadas dá-lhes espaço para evoluir e para se adaptarem, sem que o conteúdo se torne ilegível para o utilizador (Gustafson 2011). É graças a este conceito de tolerância a falhas que se torna possível aceder a um *website* construído em HTML5 e ainda utilizar CSS3 em Internet Explorer 8, que não suporta as versões mais recentes destas linguagens.

No entanto, esta característica dos *browsers* serem tolerantes a falhas pode provocar falhas de formatação. Dado que o *browser* é incapaz de processar as formatações que não suporta, essas formatações ficarão por aplicar, prejudicando assim a qualidade da interface e reduzindo a sua plasticidade. O *progressive enhacement* visa resolver este problema: se partirmos das formatações mais básicas, e posteriormente prosseguirmos para mais complexas (apenas compatíveis com versões mais atuais dos *browsers*), asseguramos a correta formatação dos elementos, independentemente do *browser*.

O *progressive enhacement* atua ao nível do código (HTML, CSS, JavaScript) que é fornecido ao *browser*, isto é, na interface final, adaptando o código que é aplicado conforme as condições do *browser*. A vantagem da adaptação ser apenas na interface final é que não é necessário alterar os modelos abstratos e concretos. O inconveniente é que a interface final fica tecnologicamente mais complexa, pois necessita considerar diferentes condições de adaptação.

2.2.3 Níveis de experiência

Na perspetiva do *progressive enhacement*, o principal foco é o conteúdo, e tudo é construído a partir dai. Essa construção pode ser vista como uma abordagem em camadas, que tenta ir ao encontro das necessidades do utilizador, prestando atenção às condições e ao contexto em que as páginas são acedidas, adaptando assim a experiência de utilização em conformidade com as capacidades do *browser* (ver Figura 5).

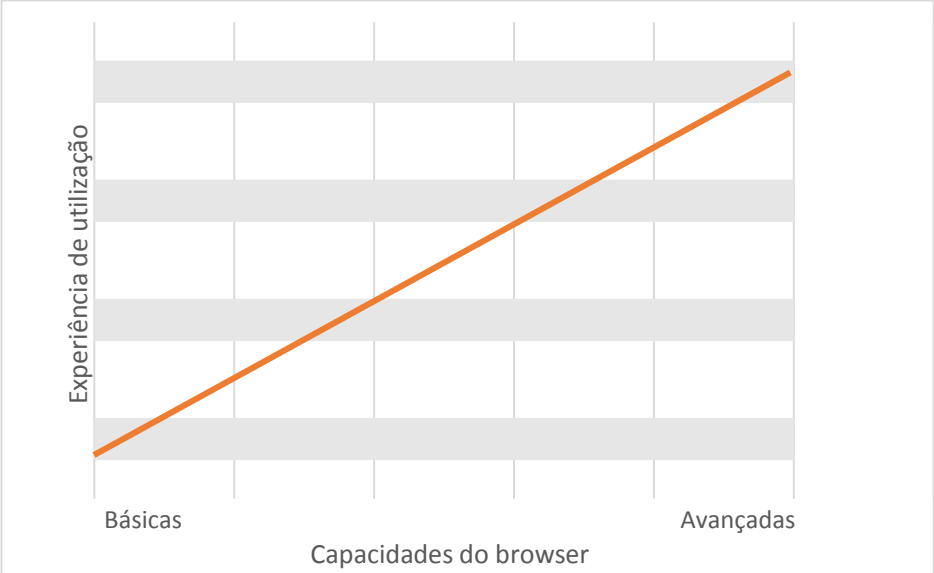


Figura 5. *Progressive enhacement* – experiência de utilização proporcional às capacidades do *browser* (Gustafson 2011)

A camada mais básica consiste somente no conteúdo em forma de texto, sem qualquer ligação a uma tecnologia específica.

O segundo nível de experiência equivale à linguagem HTML e toda a semântica que envolve. Os vários elementos e atributos providenciam um significado adicional ao conteúdo presente. A este nível acresce a importância da estrutura semântica de cada *website* ser

utilizada pelos motores de pesquisa como uma componente de indexação das páginas *web* e ainda nos resultados de pesquisa que apresenta.

O terceiro nível é marcado pelas sensações audiovisuais, onde é introduzida a linguagem CSS e a utilização de imagens, áudio e vídeo.

O quarto nível de experiência já inclui interação com o utilizador e incide sobre a introdução de JavaScript, obtendo interfaces mais dinâmicas. No entanto, a ocorrência de falhas no código desta linguagem de programação podem provocar quebras no *browser*, tornando-o assim intolerante a falhas de JavaScript. Deste modo, manter o código não-obstrutivo, isto é, sem erros que possam prejudicar a interface e a sua usabilidade, e melhorá-lo sempre que possível, são tarefas fundamentais para ajudar no *progressive enhacement*.

O nível final incide sobre possíveis melhorias e boas práticas, em conjunto com as especificações de WAI-ARIA (*Web Accessibility Initiative - Accessible Rich Internet Applications*). WAI-ARIA são especificações técnicas definidas pelo W3C para aumentar a acessibilidade das páginas *web* (W3C 2011).

Estes níveis de experiência podem ser vistos também como níveis de suporte que, encadeados, vão do nível mais básico para o mais complexo, e que vão permitir ao utilizador desfrutar de uma experiência de utilização cada vez mais enriquecida à medida que avançamos para um nível superior (Gustafson 2011). Assim, podem ser feitas melhorias à interface conforme as características da plataforma utilizada, em cada momento, pelo utilizador. Por exemplo, o suporte à navegação entre as páginas de um *site* poderá ser efetuado de forma distinta em dispositivos com diferentes tamanhos de ecrã. Desta forma, a interface torna-se capaz de suportar diferentes formas de navegação dependendo de características dos dispositivos, por exemplo utilizando JavaScript para transformar a navegação típica de um *site* num elemento *select*, como sugere a imagem da Figura 6.

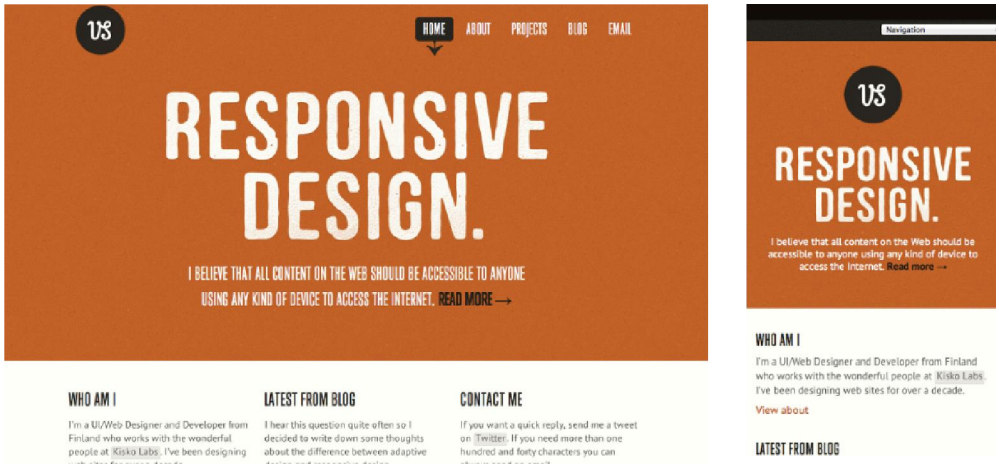


Figura 6. Exemplo da navegação de um *site* adaptada ao tamanho do dispositivo

O objetivo é melhorar a experiência de utilização dos utilizadores que acedem à *web* através dos seus dispositivos móveis. Esta transformação, ao requerer Javascript, reside no quarto nível de experiência. Se este nível não estiver disponível, no caso deste exemplo concreto é ainda possível adaptar a formatação da navegação (utilizando CSS – terceiro nível de experiência) para garantir a correta apresentação do menu em ecrãs mais pequenos. No entanto, a experiência de utilização oferecida será inevitavelmente inferior.

A navegação adequada às condições de utilização do *site* é um dos fundamentos para o *Responsive Web Design*. Segundo Frost (2012) *Responsive Web Design* é um subtópico de *Adaptive Web Design* focado na flexibilidade das interfaces que respondem às alterações do tamanho do ecrã.

2.3 Responsive Web Design

Responsive web design está relacionado com o conceito de desenvolver *websites* que sejam capazes de se adaptar ao tamanho do ecrã do dispositivo utilizado, e requer a utilização de três noções fundamentais para a sua implementação: (1) *layouts* flexíveis, baseados em *grids*, com dimensões relativas; (2) conteúdos de imagem e vídeo flexíveis, através de redimensionamento dinâmico; e por fim, (3) *media queries* e *media query listeners* (Marcotte, 2011).

Media queries são expressões que verificam determinadas condições da página, aplicando diferentes regras de CSS para diferentes cenários (Rivoal 2012). Neste contexto, é possível usar

uma *style sheet* para ecrãs de dimensões maiores, e uma outra diferente especialmente criada para dar suporte a dispositivos móveis. O recurso a *media queries* permite suportar diferentes resoluções e dispositivos, sem nunca haver a necessidade de modificar o conteúdo (La 2010). *Media query listeners* foram criadas pelo W3C e providenciam uma API para interpretar e responder às regras definidas nas *media queries*.

Com *responsive*, a utilização de *layouts*, que geralmente é expressa em *pixels*, passa a ser dimensionada através de percentagens, assim como o tamanho dos textos que passa a ser também escalável. O redimensionamento de imagens e outros géneros de *media* resulta numa adaptação automática destes elementos dependendo do dispositivo em que são carregados, atribuindo um dimensionamento percentual.

Esta combinação de *grids* fluidas, imagens de tamanho escalável e *media queries* criam uma forte solução para construir aplicações *responsive* dinâmicas e fluidas (Graeve 2011).

2.3.1 Grid Flexível

Uma *grid* é uma estrutura bi-dimensional usada para estruturar conteúdo, que subdivide uma página verticalmente e horizontalmente em margens, colunas e blocos de conteúdo.

O sistema de *grids*, que começou por emergir nas páginas de livros e jornais, forma atualmente a *layouts* de páginas *web*.

Na Figura 7 está saliente a estrutura que suporta a página HTML ai representada, que é um sistema de *grids*, onde é percetível a disposição dos conteúdos conforme essa mesma estrutura.

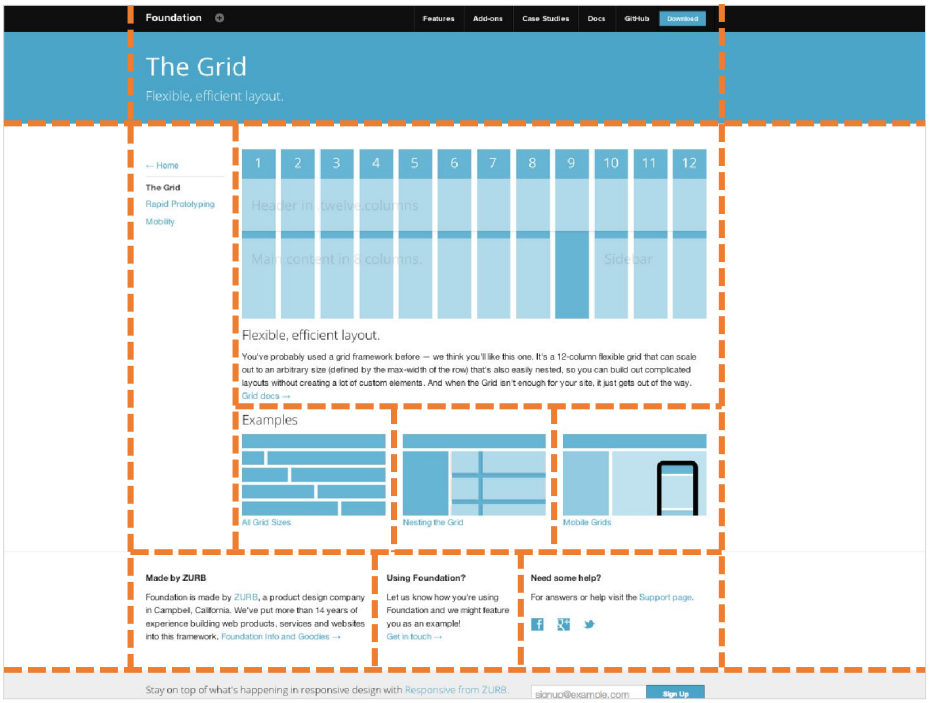


Figura 7. Exemplo de um sistema de *grids*[3]

Segundo os autores mais céticos, a utilização de *grids* inibe ou limita a criatividade dos *designers*. Por outro lado, outros autores defendem que o uso de *grids* torna mais rápido o *design* das páginas, e garante ainda a coerência visual entre páginas relacionadas, principalmente quando temos *website* com várias páginas (Boulton, 2012).

A utilização de um sistema de *grids*, aplicadas a páginas *web*, estão na origem do *responsive web design*. Com a exploração de *grids*, descobriu-se que é possível transformar a base estrutural (que até então utilizava medidas em *pixels* – Figura 8) numa componente fluida (com medidas percentuais – Figura 9).

[3] http://foundation.zurb.com/grid.php (acedido em 02/07/2013).

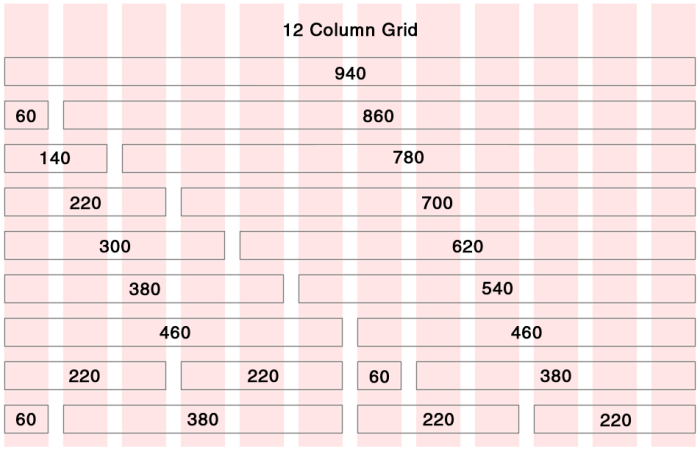


Figura 8. *Grid* fixa[4]

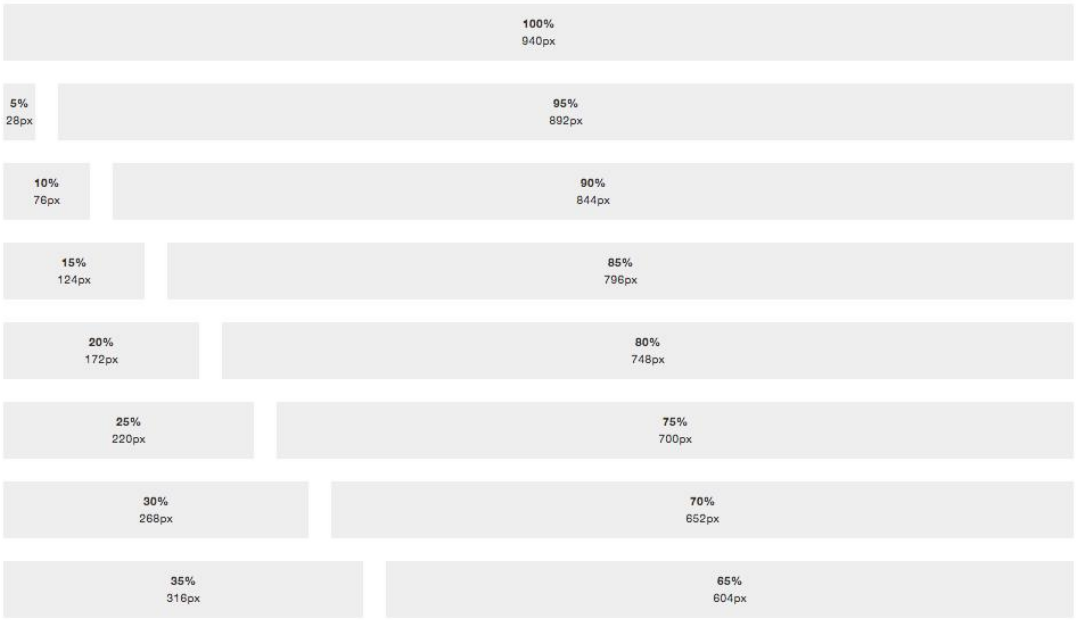


Figura 9. *Grid* fluida[5]

Para existir uma maior flexibilidade nas páginas *web*, a definição de tamanhos fixos para os elementos, que até então era feito inteiramente em *pixels*, passou a dar lugar às unidades relativas, através de medidas percentuais (%) e de medidas cujo tamanho é determinado a partir do tamanho corrente ("em" ou "rem").

[4] http://960.gs/demo.html (acedido em 15/08/2013)

[5] http://unsemantic.com/demo-responsive-rtl (acedido em 16/08/2013)

Para definir o comprimento da *grid*, passa a ser necessário ter em conta não só o tamanho que desejamos, mas também a sua relação proporcional com o tamanho do elemento que a contém. Assim, quando o elemento é redimensionado, a *grid* mantém-se proporcional ao seu tamanho. É através de CSS que se especificam todas as medidas.

Considere-se o seguinte exemplo: pretende-se definir o comprimento dos elementos "article" e "aside", para que a disposição destes elementos seja semelhante ao que vemos na Figura 10. Assume-se que se pretende um comprimento de 350px para o elemento "aside", e que o elemento que o contém, neste caso, a própria página, tem de comprimento de 988px.

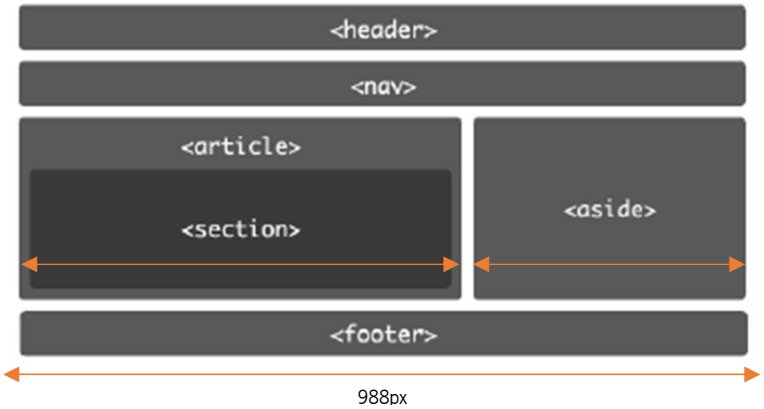


Figura 10. Estrutura de uma página HTML5

Para obter o respetivo comprimento relativo de "aside", divide-se o valor do comprimento desejado em *pixels* - *target* de 350px, pelo comprimento do elemento que o contém - *context* de 988px:

```
350 ÷ 988 = 0.354
```

O valor que se procurava é então 0.354, ou seja, 35.4%. Este valor pode ser aplicado diretamente na *stylesheet* CSS, da forma:

```
aside   { width: 35.4%; } /* 350px / 988px = 0.354 */
article { width: 64.6%; } /* 100% - 35.4%  = 64.6% */
```

Daqui pode inferir-se a seguinte fórmula, válida para determinar todos os tamanhos relativos desejados:

```
target ÷ context = result
```

Qualquer que seja o tamanho da janela de visualização garantimos que o elemento "aside" é 35.4% dessa área, e que "article" são os restantes 64.6%.

No entanto é necessário ter em consideração um cenário particular deste exemplo, a situação em que é utilizado um dispositivo com um ecrã menor, como por exemplo um telemóvel, para visualizar a mesma página. Em certas situações, o facto de existirem duas ou mais colunas de conteúdo pode dificultar a leitura do texto e até a visualização das imagens, como mostra a Figura 11.

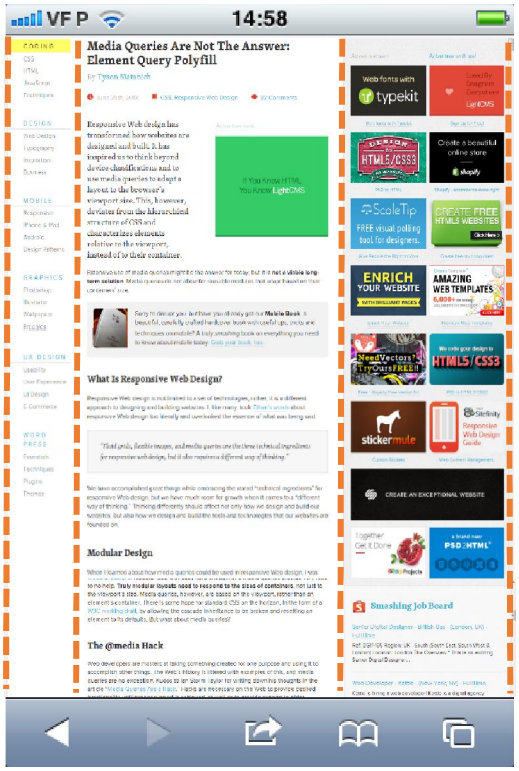


Figura 11. *Grid* com 3 colunas

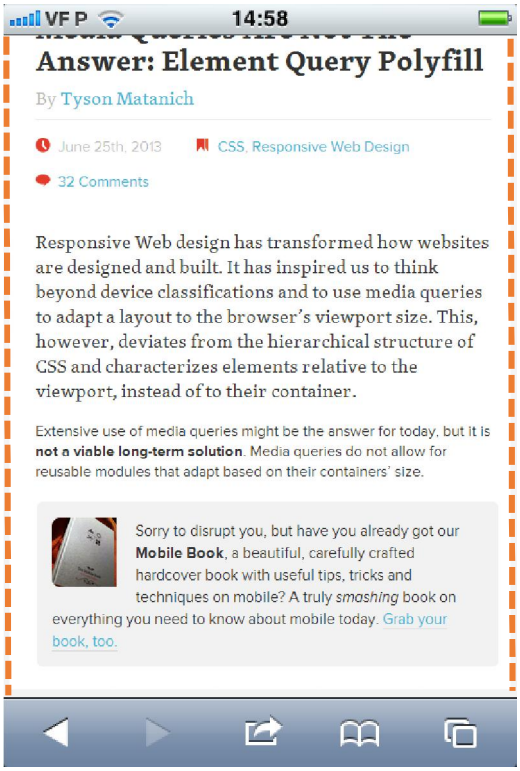


Figura 12. *Grid* com colunas sobrepostas em camadas

Dispor as colunas em camadas sobrepostas, formando uma só coluna de conteúdo, é a solução para a adaptação do conteúdo da página *web* (Figura 12). Para isso define-se que os elementos "article" e "aside" devem comportar-se com comprimento de 100% quando são detetados tamanhos de ecrã de menores dimensões (ver Figura 13).

Por vezes, no caso de existirem colunas de menor relevância, como é o caso do conteúdo de "aside" apresentado no exemplo anterior, que apenas contém publicidade, pode optar-se por simplesmente não carregar esse conteúdo.

Figura 13. Estrutura de uma página HTML5 num dispositivo móvel

Esta alternativa (Figura 13) corresponde ao comportamento *default* dos sistemas de *grids* fluidos desenvolvidos nos dias de hoje. Existem algumas *grids* pré-concebidas, facilmente acessíveis e que aceleraram a construção do site. Embora não seja flexível, o 960 Grid System (Figura 8) é um dos mais populares. Como sistemas de *grids* flexíveis podem destacar-se Gridpak[6], que permite gerar *grids* com medidas customizadas e Unsemantic[7], que está ilustrado na Figura 9.

2.3.2 Imagens, vídeos e tipografia fluidos

2.3.2.1 Imagens fluidas

A ideia por trás das imagens fluidas consiste em manter as imagens no tamanho máximo a que estas poderão ser usadas. Não são definidos quaisquer valores para a largura e altura. O *browser* está assim encarregue de redimensionar o tamanho das imagens quando necessário,

[6] http://gridpak.com/ (acedido em 15/08/2013)

[7] http://unsemantic.com/(acedido em 16/08/2013)

usando CSS para orientar o tamanho relativo de cada uma (Figura 14). Como resultado, as imagens apresentam as suas dimensões originais sempre que a largura não exceda a largura do contexto onde se inserem. Caso contrário, são redimensionadas. Os *browsers* atuais são já capazes de manter a resolução das imagens intacta, mesmo que as imagens se apresentem com uma escala diferente. A adaptação das imagens ao tamanho do elemento em que se encontra consegue-se aplicando um `max-width` de 100%:

```
img { max-width: 100%; }
```

Para vídeos embebidos pode ser aplicada a mesma solução:

```
img, object { max-width: 100%; }
```


Figura 14. Elementos escaláveis em RWD

Esta é a forma mais simples para transformar imagens de tamanho fixo em imagens fluidas – deixar para o *browser* a responsabilidade de escalar as imagens. No entanto, nem sempre resulta: as versões antigas do Internet Explorer não suportam `max-width`. Para contornar este problema nas versões antigas de Internet Explorer, e se as imagens forem de suficiente resolução, pode ser uma opção considerar:

```
img { width: 100%; }
```

No entanto, isto implica uma condição específica na *stylesheet* para Internet Explorer, porque caso contrário, as imagens teriam sempre o máximo tamanho possível, e isso não é o que se pretende.

Uma outra alternativa seria recorrer a JavaScript para contornar o problema e redimensionar a imagem depois de descarregada (Marcotte 2010). No entanto, o redimensionamento das imagens através do *browser* pode ser um processo demasiado intrusivo para dispositivos com capacidades mais limitadas.

É nesta questão das imagens fluidas que se reúnem a maioria das críticas em RWD. O problema reside na utilização imagens com alta resolução, que são carregadas pelos *browsers* mesmo quando os utilizadores têm uma reduzida largura de banda disponível. A ideia de que carregar imagens com alta ou baixa resolução deve ser uma opção do utilizador é defendida por (Hannemann 2013). Recentes trabalhos do W3C referem uma extensão HTML `<picture>` para imagens adaptativas, capaz de servir diferentes imagens com diferentes tamanhos, conforme o tamanho do dispositivo utilizado (W3C 2013).

Atualmente existem algumas ferramentas que funcionam como alternativas para obter imagens fluidas tentando contornar os inconvenientes associados à utilização de imagens com grande resolução.

2.3.2.2 Ferramentas de imagens e vídeo fluidos

Como vimos anteriormente, um elemento essencial para um *design responsive* é a presença de imagens com características fluidas, isto é, com a capacidade de se ajustarem relativamente ao tamanho do contexto onde se encontram. No entanto, esta componente parece trazer alguns desafios em relação à capacidade de otimização de recursos e durante o carregamento das páginas *web*, com principal impacto nos dispositivos móveis. Se por um lado as imagens *desktop-optimized* têm demasiada resolução para conexões móveis, e as imagens *mobile-optimized* têm uma qualidade inferior à necessária para se obter bons resultados visuais em computadores.

Para este caso, é importante perceber se existem ferramentas que sejam capazes de contornar este problema. De seguida são descritas as ferramentas que mais se destacam e que

lidam com este problema, devolvendo a imagem com o tamanho mais apropriado conforme o tamanho do ecrã do dispositivo que está a ser utilizado.

- **Adaptive Images**[8]

Matt Wilcox criou Adaptive Images, uma componente PHP que deteta o tamanho do ecrã, e posteriormente cria, armazena em cache e devolve automaticamente uma versão da imagem com o tamanho mais apropriado. Na Figura 15 estão exemplos da sua utilização.

É uma ótima solução para *sites* onde que não há a capacidade de reestruturar o código, uma vez que não necessita de utilizar *markup* extra, apenas incluir os ficheiros necessários.

Existem alguns parâmetros customizáveis, como por exemplo o tamanho das imagens que vão servir diferentes intervalos de tamanhos de ecrã:

```
$resolutions = array(860, 600, 320);
```

O valor mais pequeno, neste caso 320px, é o tamanho da imagem que será criada para os ecrãs que não excedem esse comprimento. Por exemplo, um ecrã com 300px será servido com uma imagem com a resolução de 320px, porque este é o valor mínimo definido no *array* `$resolutions`. Se, por outro lado, o ecrã for de 321px, como esse valor excede os 320px, será devolvida a imagem com o tamanho seguinte, que neste caso é 600px.

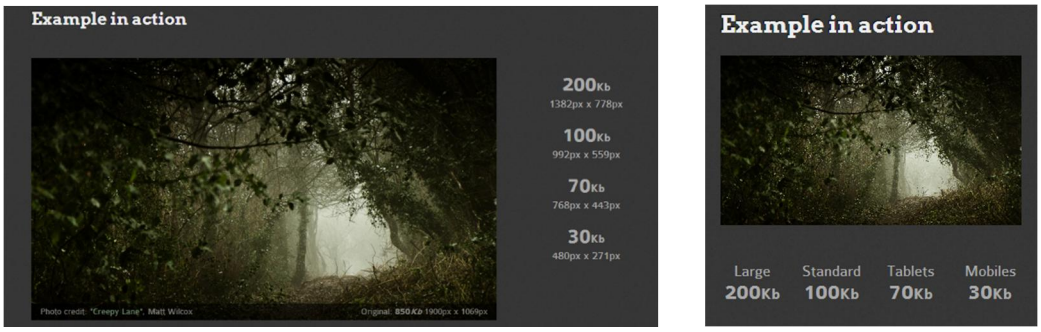


Figura 15. Exemplo de utilização do Adaptive Images: vista a partir de um *laptop* (à esquerda) e *smartphone* (à direita)

[8] http://adaptive-images.com/ (acedido em 03/07/2013)

- **Sencha.io Src**[9]

Sencha.io Src é um serviço, originalmente criado por James Pearce, que devolve versões otimizadas das imagens consoante do tamanho do dispositivo. Para utilizar é necessário indicar no atributo fonte (`src`) da imagem o endereço do Sencha.io Src, seguido do endereço da imagem, por exemplo:

```
http://src.sencha.io/http://mysite.com/images/UluruRock.jpg
```

Utiliza *user agents* para descobrir qual é o tamanho do dispositivo, redimensionando a imagem adequadamente (ver Figura 16). Por omissão a imagem é redimensiona de maneira a que a fique a ocupar 100% do comprimento do ecrã do dispositivo.

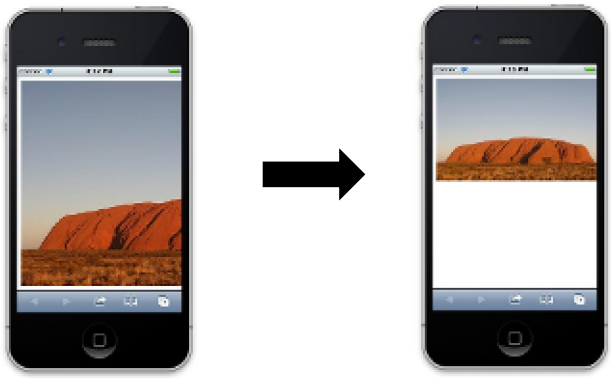


Figura 16. Exemplo de utilização do Sencha.io Src

É também possível redimensionar as imagens para um tamanho específico, adicionando um outro parâmetro com o tamanho pretendido. Por exemplo, se quisermos redimensionar a imagem com comprimento de 320px, escrevemos:

```
http://src.sencha.io/320/http://mysite.com/images/UluruRock.jpg
```

Sencha.io Src é ainda inteligente para fazer cache dos pedidos, evitando que a imagem seja gerada cada vez que a página é carregada.

[9] http://docs.sencha.io/current/index.html#!/guide/src (acedido em 03/07/2013)

Até agora as duas ferramentas apresentadas são métodos que utilizando o servidor e alguma forma de deteção para determinar a imagem que deve ser devolvida. No entanto, essa tarefa pode ser feita do lado do *browser*, como é o caso das ferramentas que se seguem.

- **Picturefill**[10]

Scott Jehl desenvolveu esta ferramenta para simular o comportamento do elemento `picture` recorrendo aos elementos compatíveis com os *browsers* atuais. Esta ferramenta é utilizada pelo *site* da Microsoft para servir imagens com diferentes resoluções conforme o tamanho do ecrã do dispositivo utilizado.

- **FitVids**[11]

Se utilizarmos a *tag* de vídeo em HTML5, basta usar a técnica max-width:100% para conseguirmos tornar este elemento fluido. No entanto, vemos frequentemente a utilização de *sites* externos como YouTube e Vimeo, que utilizam um iFrame para manter os vídeos embebidos nas páginas *web*. Se usarmos a mesma técnica, o vídeo escala em comprimento mas não em altura, quebrando a sua proporcionalidade.

O FitVids é um *plugin* jQuery (desenvolvido por Chris Coyier, Trent Walton, Dave Rupert e Reagan Ray) leve e simples de usar, em que os vídeos respondem a diferentes tamanhos do ecrã mantendo sempre a sua relação de proporção original. Um exemplo da sua utilização pode ser visto na Figura 17.

[10] https://github.com/scottjehl/picturefill (acedido em 15/08/2013)

[11] http://fitvidsjs.com/ (acedido em 12/07/2013)

Figura 17. FitVids: vista a partir de um *laptop* (à esquerda) e *smartphone* (à direita)

2.3.2.3 Tipografia fluida

Redimensionar o texto consoante o tamanho do dispositivo é fulcral para proporcionar uma boa experiência de leitura aos utilizadores. O CSS3 introduziu novas medidas tipográficas baseadas no tamanho do *viewport* (consultar a Tabela 1). As unidades de *viewport-percentage* são relativas ao tamanho inicial do contexto onde o texto se encontra. Quando o comprimento e/ou a largura do elemento que o contém é alterado, o tamanho da fonte é redimensionado. (W3C 2013)

Tabela 1. Unidades de percentagem do *viewport*

Unidade	Descrição
vw	(*viewport width*) tamanho do *viewport*
vh	(*viewport height*) altura do *viewport*
vmin	valor de "vw" ou "vh", dependendo de qual for o menor
vmax	valor de "vw" ou "vh", dependendo de qual for o maior

Considere-se a seguinte definição do tamanho de letra do elemento "h1":

```
h1 { font-size: 4vw; }
```

A unidade vw (*viewport width*) é igual a 1/100 ou 1% do tamanho do *viewport*. Neste exemplo, se o comprimento do *viewport* é de 1000px, o tamanho da fonte do elemento `h1` será de 40px. Isto porque 4×1000/100 = 40.

Encolhendo o tamanho da janela do *browser* para 480px de comprimento, o tamanho do elemento `h1` reduz-se em simultâneo para 19px (4×480/100 = 19.2).

É possível obter-se resultados semelhantes utilizando as unidades relativas já existentes no CSS2. No entanto, o que torna estas novas unidades tão promissoras é a sua total independência dos elementos hierarquicamente ligados. Ao contrário de unidades como "em", cujo valor depende sempre do valor do tamanho de letra do elemento hierarquicamente superior, as unidades de percentagem do *viewport* são unicamente dependentes do tamanho do *viewport* atual (Sampaio 2013).

2.3.2.4 Ferramentas de tipografia fluida

- **FitText**[12]

FitText é um *plugin* desenvolvido em jQuery que expande ou retrai o texto de forma a preencher todo o espaço de conteúdo, criando um efeito semelhante ao que resulta da utilização de unidade de *viewport*. Qualquer que seja o tamanho do ecrã, o texto ocupa todo o comprimento disponível.

É apresentado um exemplo da sua utilização (código e resultado obtido) na Figura 18.

```
<script type="text/javascript">
     $("h1").fitText(1.2);
     $("h2").fitText(1.5);
     $("p").fitText(1.0, {
      minFontSize: '12px',
      maxFontSize: '16px' });
</script>
```

Figura 18. FitText: exemplo de utilização

Uma outra ferramenta muito semelhante é o FlowType[13], que determina e apresenta o tamanho da fonte conforme o comprimento do elemento.

[12] http://fittextjs.com/ (acedido em 12/07/2013)

- **SlabText**[14]

SlabText é um *plugin* jQuery, que permite criar grandes títulos e blocos de texto fluido, adaptando-se ao espaço disponível do elemento que o contem, dividindo em várias linhas se for necessário.

De seguida é apresentado um exemplo da utilização deste *plugin*, com o respetivo resultado final ilustrado na Figura 19.

```
<script type="text/javascript">
  function slabTextHeadlines() {
     $("h1").slabText();
};
</script>
```

Figura 19. SlabText: exemplo de utilização

2.3.3 Media queries

Media queries são um mecanismo desenvolvido pelo W3C para identificar não só os tipos de *media* com os quais se está a lidar, mas para inspecionar as características físicas dos dispositivos e *browsers*. Com a utilização destas *media queries* é possível direcionar o *design* para um conjunto específico de dispositivos, sem que seja necessária a alteração do seu conteúdo.

Uma *media query* consiste numa media *type* e zero ou mais expressões que verificam as condições de *media features* (W3C Recommendation, 2012). Alguns exemplos de *media features* são *width* (largura do elemento), *height* (altura do elemento), *color* (cor), *device-width* (largura do dispositivo), *device-height* (altura do dispositivo), *orientation* (orientação do

[13] http://simplefocus.com/flowtype/ (acedido em 25/08/2013)

[14] https://github.com/freqDec/slabText/ (acedido em 12/07/2013)

dispositivo) *e resolution* (resolução do ecrã) – na Tabela 3 é apresentada uma descrição das *media features* mais relevantes para este contexto de *Responsive Web Design*.

2.3.3.1 Media types e media features

Uma característica essencial das *style sheets* é o facto de estas especificarem como é que o nosso documento tem que ser apresentado nos diferentes tipos de *media*: num ecrã, em papel, ou mesmo num dispositivo táctil em braille. Quando queremos referir-nos a um dado tipo de media utilizamos *media types*. A Tabela 2 descreve os *media types* existentes, definidos pelo W3C.

Tabela 2. Media types

Media type	Descrição
all	Refere-se a todo o tipo de dispositivos
braille	Refere-se a dispositivos tácteis em braille
embossed	Refere-se a páginas impressas em braille
handheld	Refere-se a dispositivos portáteis, geralmente pequenos e com largura de banda limitada
print	Refere-se a impressoras
projection	Refere-se a apresentações projetadas, como slides
screen	Refere-se a ecrãs de computadores
speech	Refere-se a dispositivos de som
tty	Refere-se a media com exibição limitados, como terminais
tv	Refere-se a ecrãs de televisões

Os *media types* permitem criar regras dirigidas para um determinado tipo de *media* ou dispositivo. Já as *media features* permitem inspecionar propriedades específicas do dispositivo e assim orientar as regras aplicadas conforme essas características. Combinamos *media types* e *media features* para construir um dos elementos fundamentais no RWD – as *media queries*.

Tabela 3. *Media features*

Media feature	Descrição
aspect-ratio	Corresponde ao valor do rácio entre a largura e a altura da janela de visualização
device-aspect-ratio	Corresponde ao valor do rácio entre a largura e a altura do dispositivo
device-height	Corresponde à altura do dispositivo
device-width	Corresponde à largura do dispositivo
height	Corresponde à altura da janela de visualização
orientation	Corresponde à orientação do dispositivo (*portrait* ou *landscape*)
resolution	Corresponde à resolução do dispositivo
width	Corresponde à largura da janela de visualização

2.3.3.2 Declaração de media queries

Considere-se o seguinte exemplo de uma *media query*:

```
@media screen and (min-width: 1024px) {
     body { font-size: 100%; }
}
```

Cada *media query* é iniciada pelo *media type* selecionado, que neste exemplo é `screen`. Segue-se a *query* propriamente dita, que é declarada entre parêntesis, `min-width: 1024px`. A *query* combina dois componentes: o nome da característica, `min-width`, e o seu valor 1024px.

A *media query* que compõe o exemplo interroga o *browser* sobre 1) se este se trata de um ecrã (*media type* `screen`) e, em caso afirmativo, 2) se a janela de visualização (neste caso, o *viewport* do *browser*) tem pelo menos 1024px de comprimento. Se o *browser* passar nestes dois critérios, então os estilos definidos dentro da *media query* são aplicados. Caso contrário, o *browser* ignora essas definições.

À medida que o *layout* escala e os seus elementos se redimensionam, o uso das media *queries* pode corrigir as imperfeições visuais que vão surgindo. Além disso, as *media queries* podem auxiliar na otimização do conteúdo do *site* para ir de encontro com as necessidades de

cada dispositivo, criando *layouts* alternativos para diferentes gamas de resolução (Marcotte 2011).

A estratégia utilizada para detetar que *layout* deve ser apresentado consiste em determinar o valor do *viewport*, para controlar o tamanho da área de visualização do *browser* disponível, definindo que o seu tamanho corresponde ao tamanho do ecrã `(width=device-width)`. Para isso deve definir-se no cabeçalho HTML o seguinte:

```
<meta name="viewport" content="width=device-width">
```

Podemos assim aplicar regras de acordo com o tamanho do *viewport*, consoante o valor de `width`, `min-width` e `max-width`:

```
@media screen and (width: 320px)       { … }
@media screen and (max-width: 480px)   { … }
@media screen and (min-width: 1200px)  { … }
```

Existem ainda formas automática de deteção do dispositivo utilizado, baseada na informação providenciada por *user agents*. É o caso da ferramenta Categorizr, capaz de detetar o dispositivo que está a ser utilizado, dividindo em quatro categorias: televisão, computador, *tablet* e telemóvel (Jankord 2012).

Media queries consistem numa *feature* de CSS3, e por isso as versões dos *browsers* que não são compatíveis com CSS3 não suportam *media queries*. Estes *browsers* ignoram as regras definidas em *media queries*, podendo causar falhas de formatação dos elementos e, consequentemente, incoerências nas interfaces. Existem, no entanto, ferramentas como Respond.js (Filament Group 2012) e css3-mediaqueries.js[15] que conseguem ultrapassar esta limitação dos *browsers*, interpretando *media queries*.

[15] https://code.google.com/p/css3-mediaqueries-js/ (acedido em 23/07/2013)

2.4 Responsive Web Design boilerplates

Boilerplates são ferramentas de *web design* que contém elementos HTML pré-fabricados e formatados. São portanto um acelerador para o desenvolvimento da interface de um *website* ou aplicação *web*. Estas *frameworks* combinam os elementos essenciais para um *design responsive*, e algumas delas combinam mesmo o melhor das ferramentas mencionadas na Secção 2.3 numa só.

Foi feito um estudo sobre quais as ferramentas mais utilizadas atualmente e as que merecem maior destaque. De seguida são apresentadas as conclusões obtidas a partir deste estudo, dentro dos *boilerplates* de maior relevância.

- **Gridless**

Gridless pode ser visto como um ponto de partida para um projeto *web*, baseado numa abordagem *mobile first*. Não traz quaisquer classes semânticas pré-definidas e, ao contrário da maioria dos *boilerplates*, não inclui um sistema de *grids,* dando ao utilizador a flexibilidade de usar o seu próprio. Estas características do Gridless facilitam a sua adaptação a qualquer projeto. A Figura 20 mostra um exemplo da utilização desta *framework*.

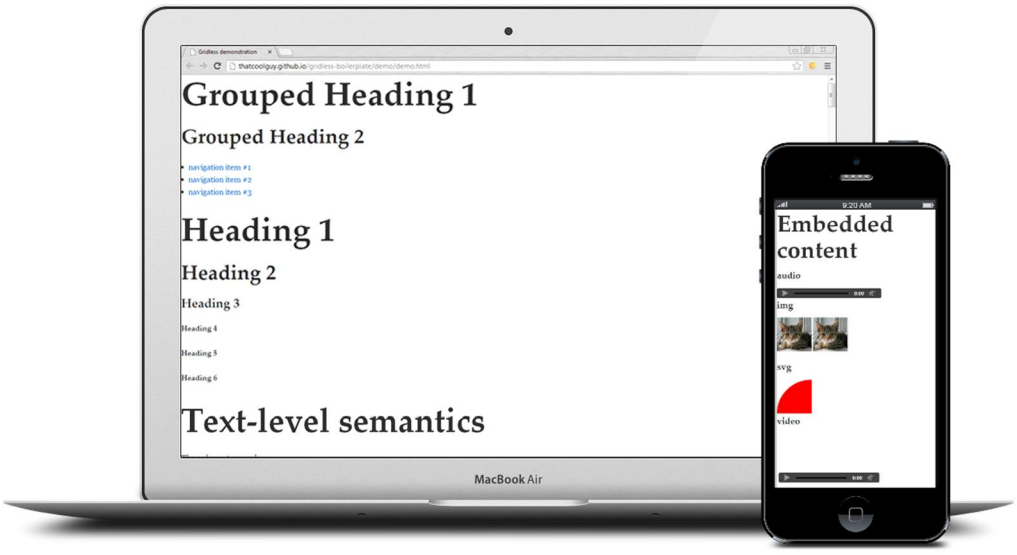


Figura 20. Exemplo de utilização da *framework* Gridless[16]

[16] http://thatcoolguy.github.io/gridless-boilerplate/demo/demo.html (acedido em 19/10/2013)

O Gridless foca-se na compatibilidade com todos os *browsers*. Por isso, foi construído com base no princípio de *progressive enhacement* e Integra Respond.js (Secção 2.1.3) como ferramenta auxiliar para garantir o suporte de *media queries* mesmo nos *browsers* que não suportam CSS3.

- **Skeleton**

Skeleton é um *kit* de desenvolvimento, baseado no 960.gs *grid system*, que escala a partir dos ecrãs de maior resolução para os de menor resolução (uma abordagem *desktop-first*). Incorpora estilos básicos e está preparado para ser adaptado a qualquer aplicação *web*. Um exemplo da sua utilização encontra-se na Figura 21.

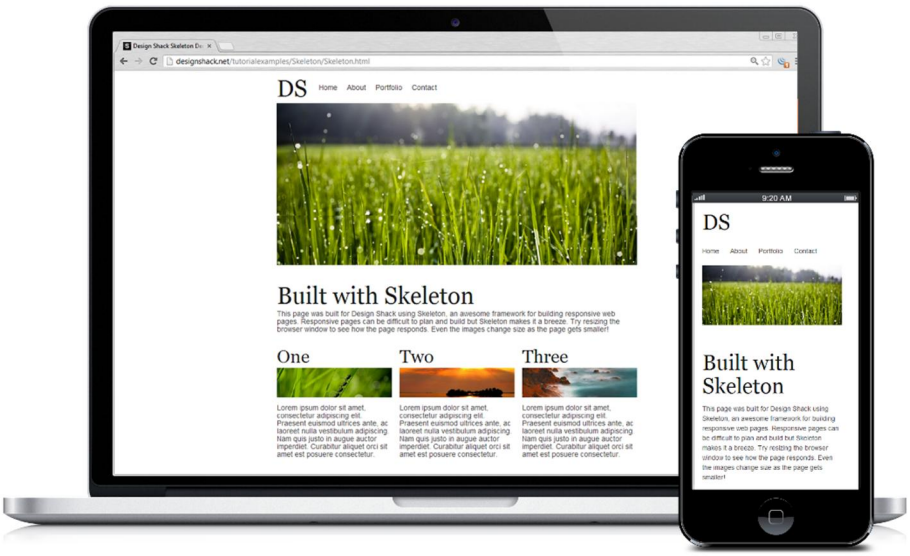


Figura 21. Exemplo de um *site* construído com a *framework* Skeleton[17]

- **Zurb Foundation**

Desenvolvido pela Zurb, é uma *framework* para desenvolvimento de interfaces bastante poderosa na atualidade. A figura 22 mostra um *site* construído com esta base nesta *framework*. Foundation inclui o seu próprio sistema de *grids* flexível, e ainda um conjunto de componentes e classes semânticas que facilitam e aceleram a prototipagem e desenvolvimento da interface de uma aplicação *web*.

[17] http://designshack.net/tutorialexamples/Skeleton/Skeleton.html (acedido em 19/10/2013)

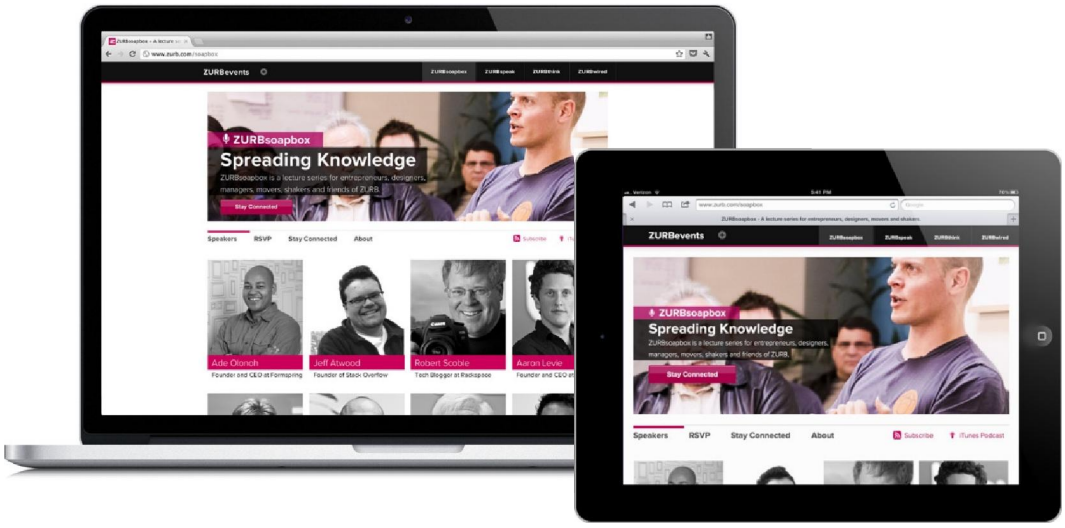


Figura 22. Exemplo de um *site* construído com a *framework* Foundation[18]

- **Twitter Bootstrap**

Desenvolvido pelo Twitter, é uma poderosa ferramenta para criação de interfaces que providencia um rápido e fácil desenvolvimento *web*. Inclui um conjunto de 12 *responsive grids* e dezenas de componentes e classes pré-definidas. Possibilita ainda a customização dos seus estilos base, facilitando a adaptação desta ferramenta a cada projeto. A Figura 23 contém um exemplo de *site* desenvolvido com base no Bootstrap.

[18] http://foundation.zurb.com/case-soapbox.php (acedido em 19/10/2013)

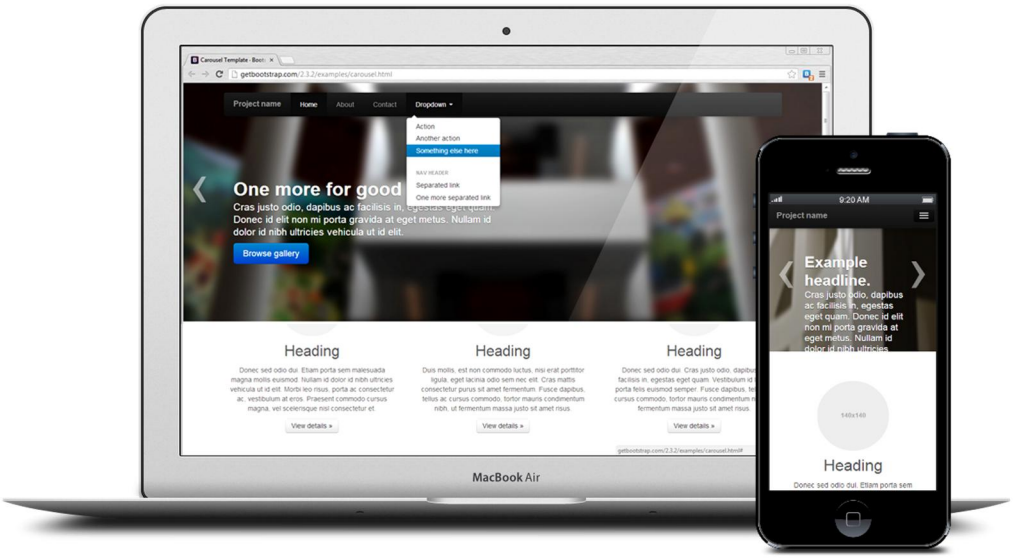


Figura 23. Exemplo de um *site* construído com a *framework* Bootstrap[19]

- **Pure**

Pure é uma ferramenta lançada em 2013 pela Yahoo que pretende servir como base de formatação para qualquer aplicação *web*. Integra a biblioteca YUI, também desenvolvida pela Yahoo, como auxiliar para o sistema de *grids*. A Figura 24 ilustra um *site* exemplo, desenvolvido com base em Pure. Embora possua menos componentes quando comparada com Foundation ou Bootstrap, esta ferramenta está preparada para ser facilmente extensível a todos os componentes fornecidos pela YUI.

Outras características muito convenientes desta ferramenta é o facto de ser muito leve e modular, ou seja, é possível incluir apenas certos módulos, conforme for necessário. Incorpora uma ferramenta de customização *online*, para facilitar a adaptação e integração com qualquer projeto *web*.

[19] http://getbootstrap.com/2.3.2/examples/carousel.html (acedido em 19/10/2013)

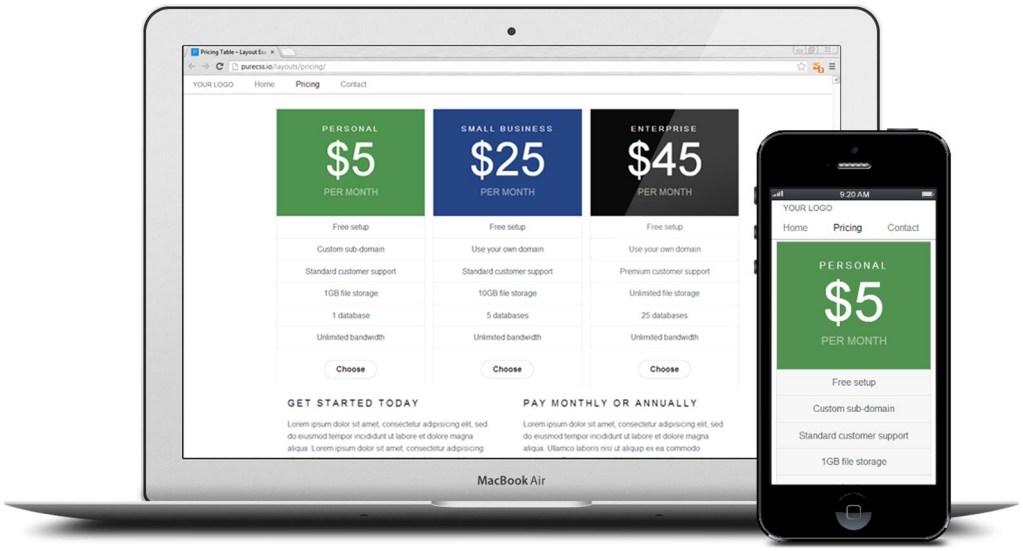


Figura 24. Exemplo de um *site* construído com a *framework* Pure[20]

Para além destas existem outras ferramentas como a Ink[21] (desenvolvida pela empresa Portuguesa Sapo) e a Kube[22], que se assemelham ao Bootstrap tanto nos elementos pré-fabricados que oferecem, como pelo facto de seguirem uma abordagem *desktop-first.*

Bootstrap e Foundation são *frameworks* bastante poderosas, que diferem essencialmente em dois aspetos principais. O primeiro consiste na linguagem de *style sheet* que utilizam: ainda que ambos utilizem uma metalinguagem de CSS, Foundation é baseado em Sass, enquanto Bootstrap foi construído em Less. Sass e Less são pré-processadores de CSS que permitem estender esta linguagem, suportando, por exemplo, a existência de variáveis, funções e operações básicas sobre cores e medidas. Apresentam apenas ligeiras diferenças em termos de sintaxe, mas diferem na forma como são processadas. Less, por ser uma biblioteca em JavaScript, é processada do lado do cliente, ao passo que Sass foi desenvolvida em Ruby on Rails, e por isso é processada do lado do servidor (Hixon 2011).

O segundo aspeto centra-se no tipo de abordagem de desenvolvimento que segue cada uma destas ferramentas. Foundation segue uma abordagem *mobile-first*, ao contrário do

[20] http://purecss.io/layouts/pricing/ (acedido em 19/10/2013)

[21] http://ink.sapo.pt/ (acedido em 16/08/2013)

[22] http://imperavi.com/kube/ (acedido em 16/08/2013)

Bootstrap. Esta abordagem do Bootstrap baseada em *desktop-first* pode tornar-se uma desvantagem, devido ao facto de a adaptação para ecrã com menor resolução ser mais difícil de alcançar utilizando esta *framework*, isto porque os seus componentes podem não ser totalmente escalável a qualquer dispositivo.

O gráfico da Figura 25 mostra a evolução das pesquisas no Google por parte dos utilizadores nos últimos 2 anos – desde junho de 2011, ano em que o tema *responsive web design* se tornou mais popular na comunidade web, até junho de 2013 – utilizando a ferramenta Google Trends.

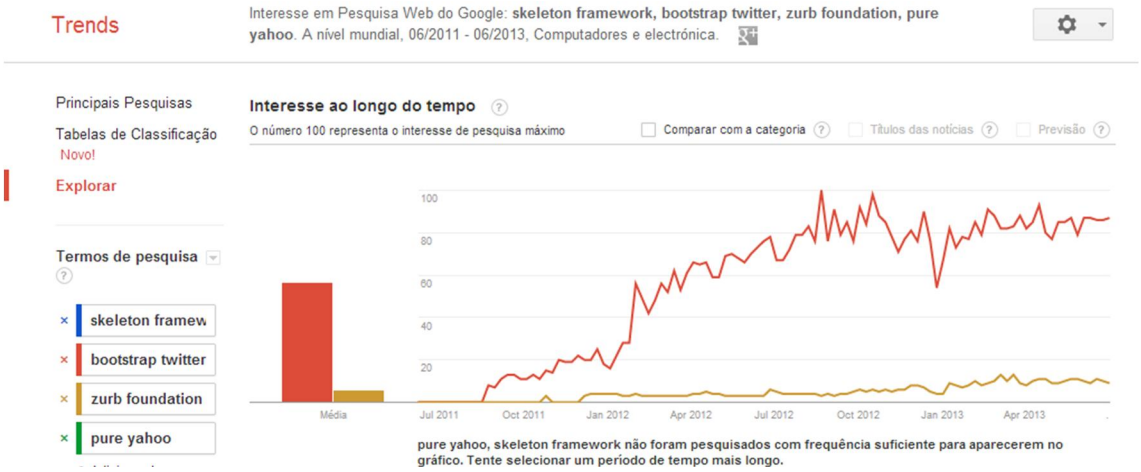


Figura 25. Tendências de utilização de responsive *boilerplates* (Google Trends)

Quando comparadas com Bootstrap e Foundation, as restantes ferramentas não têm qualquer relevância nas pesquisas dos utilizadores, como é percetível a partir do gráfico. A popularidade e confiança depositada no Twitter estão com certeza ligados com a esta esmagadora tendência a favor do Bootstrap. Também a Zurb, por ser uma empresa conceituada no desenvolvimento e *design web*, poderá ter incentivado a comunidade a experimentar e usar a *framework* Foundation. A crescente popularidade da linguagem de programação Ruby on Rails pode também ter levado à preferência dos *developers* por esta *framework*. Dado que a Zurb utiliza Rails no desenvolvimento das suas aplicações, a integração do Foundation é facilitada com projetos baseados em Rails. Embora a Yahoo! seja também uma empresa bastante popular, o Pure é uma ferramenta bastante recente, e por isso é expectável que ainda não seja possível a sua visualização no gráfico.

Capítulo 3

Adaptação e otimização em responsive web design

Responsive web design é um princípio com o objetivo de criar uma experiência de utilização que se adapta às necessidades dos utilizadores, e ao mesmo tempo, às capacidades e limitações dos dispositivos. Para certos tipos de adaptação (que podem ser relativos tanto ao conteúdo como aos diversos componentes que compõem um *site*) existem um conjunto de soluções padrão. Foi realizado um estudo em torno dessas soluções e, ao longo deste capítulo, serão apresentados possíveis padrões de adaptação para o *layout* e ainda para dois componentes específicos, tabelas e navegação.

Para além da questão da adaptação conforme o tamanho da janela do utilizador, é também importante a otimização dependendo do contexto do utilizador. Enquanto a adaptação se torna fundamental para que se consiga aceder à informação, a otimização pode ser vista como um complemento para aproximar a experiência de utilização à plataforma utilizada (por exemplo, otimização para dispositivos cuja interação é feita através do toque). Se até agora *sites* e aplicações *web* eram tipicamente construídas com uma única forma de acesso em mente – um computador *desktop* ou *laptop* –, nos dias de hoje o acesso à *web* pode ser feito a partir de diferentes dispositivos, com características distintas. Na segunda parte deste capítulo é descrito de que forma é possível otimizar de acordo com o contexto do utilizador, com o objetivo de garantir sempre uma boa experiência de utilização.

3.1 Padrões de adaptação

A atenção concedida à capacidade adaptativa dos *websites* levou a que a fossem criados padrões de adaptação, formas inovadoras de desenhar e desenvolver *layouts* e componentes que respondam a tais requisitos. Deste modo, é possível observar-se a adoção de alguns padrões de adaptação comuns, tanto nos *layouts* como em outros componentes como tabelas e navegações. Ao longo desta secção serão explorados alguns desses padrões. Procurar-se-á

perceber de que forma é possível implementá-los e apresentar as suas vantagens e desvantagens.

3.1.1 Layouts

Começando pela adaptação de *layouts*, o *layout* mais básico possível é constituído por uma só coluna de conteúdo flexível, que se adapta conforme o tamanho da janela de visualização utilizada (ver Figura 26). Esta é a adaptação mais simples que podemos encontrar. Neste padrão o conteúdo é definido dentro de uma estrutura 100% fluida (with:100%), sendo apenas necessário reduzir o tamanho de imagens (max-with:100%) e, eventualmente, o tamanho das fontes, à medida que o tamanho de ecrã disponível vai diminuindo.

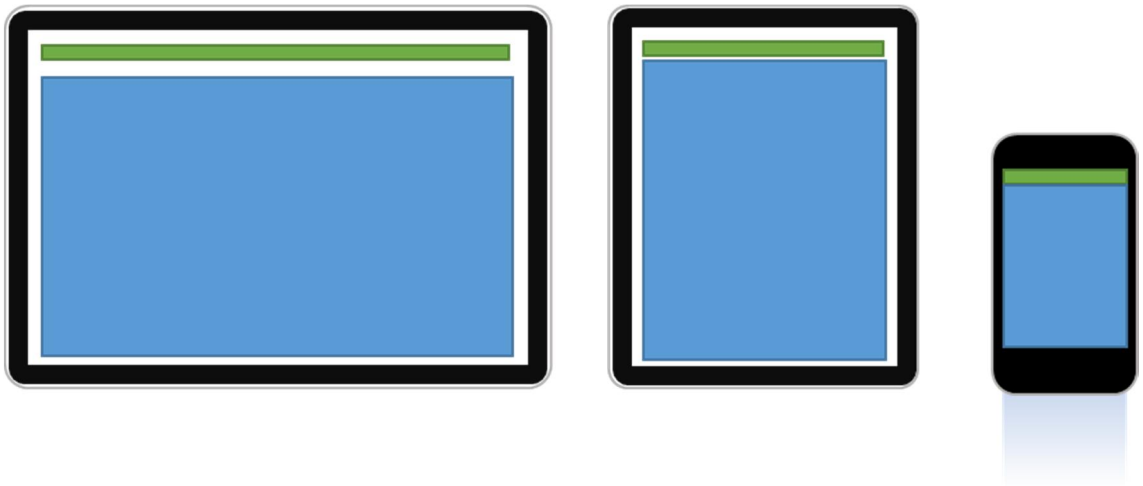


Figura 26. Adaptação de um *layout* com uma só coluna

No caso de se utilizar não uma, como no exemplo anterior, mas duas ou mais colunas de tamanho igual, para assegurarmos a total flexibilidade do conteúdo e, simultaneamente, a proporção das colunas, é necessário intervir no sentido de reestruturar a forma como estão distribuídas as várias colunas de conteúdo. Assim, garantimos que o conteúdo está disponível independentemente do tamanho do dispositivo que é utilizado (e sem recorrer à barra de *scroll* horizontal).

Caso sejam duas colunas, mantemos a estrutura de ambas com a medida de 50% do total do ecrã (width:50%) para ecrãs maiores, e reposicionamos a colunas da direita para baixo em ecrãs mais pequenos (width:100% para ambas as colunas), como ilustra a Figura 27.

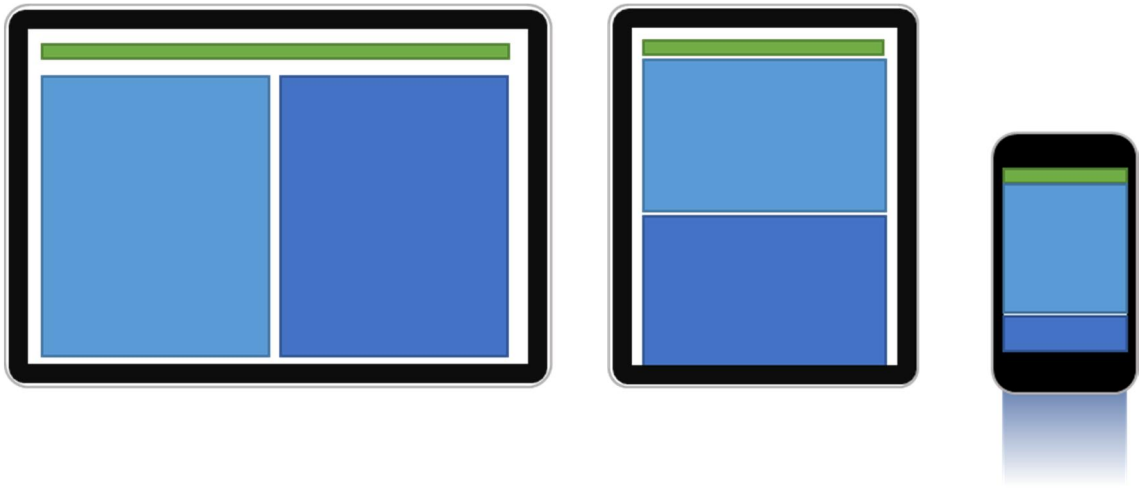


Figura 27. Adaptação de um *layout* com duas colunas

Com três colunas de conteúdo, a lógica de reestruturação é muito semelhante à do exemplo anterior. Para ecrãs maiores utilizamos colunas com a medida de 33.3(3)% relativamente ao seu tamanho total. Para ecrãs menores as colunas são empilhadas formando uma só coluna de conteúdo, tal como acontece quando são apenas duas colunas. Neste caso é utilizado o mesmo padrão adaptativo.

No caso de serem quatro colunas de conteúdo (width:25%), a adaptação pode ser feita de uma forma mais moderada, criando-se um nível intermédio onde o conteúdo é redistribuido uniformente pelo espaço disponível, formando duas colunas (width:50%). Para ecrãs mais pequenos, a solução recai novamente sobre adaptar o conteúdo a uma só coluna (ver Figura 28).

Figura 28. Adaptação de um *layout* com quatro colunas

Repare-se que uma solução com um nível intermédio para o caso de termos um *layout* com 3 colunas de igual tamanho resultaria num *layout* desequilibrado, devido à existência de um espaço vazio (provocado pela inexistência de uma quarta coluna). Por outro lado, se optássemos por distribuir a última coluna pela extensão do ecrã (width:100% apenas na coluna mais à direita), estaríamos a quebrar a proporção que existe entre as três colunas de conteúdo.

Um dos *layouts* mais comuns consiste numa coluna de conteúdo principal, com uma barra lateral localizada à direita. Neste caso, a adaptação consiste em mover a barra lateral para baixo da coluna de conteúdo. Em dispositivos de pequenas dimensões faz sentido apresentarmos em primeiro lugar o conteúdo principal, e só depois o conteúdo da barra lateral.

O mesmo acontece quando lidamos com duas barras laterais (esquerda e direita) – o conteúdo principal deve ter sempre prioridade. No entanto, se a barra esquerda assume um papel mais importante do que a do lado direito (por exemplo, para efeitos de navegação), é possível identificar um outro padrão de adaptação, ilustrado na Figura 29.

Figura 29. Adaptação de um *layout* com barras laterais

Neste padrão o conteúdo das barras laterais vai-se distribuindo à medida que o ecrã é reduzido. É mantida a barra lateral esquerda (wifth:25%) ao mesmo nível que o conteúdo principal (wifth:75%) o mais possível, reconhecendo o facto de que o conteúdo da barra esquerda é mais importante que o da direita. À medida que o ecrã diminui é utilizado um *layout* intermédio, em que a barra lateral direita é reposicionada (ao ser aplicado width:100%). Em ecrãs mais pequenos, o conteúdo é empilhado, e a barra que inicialmente foi posicionada à esquerda é movida para o local imediatamente após o conteúdo principal. Para se conseguir obter este resultado, é necessário que a definição dos três elementos na DOM tenha sido feita pela seguinte ordem: conteúdo principal, barra lateral esquerda, e por fim, barra lateral direita.

Embora seguindo estratégias de adaptação distintas, os padrões que vimos anteriormente resultam numa adaptação em que o conteúdo é empilhado em ecrãs de menores dimensões, obtendo-se por isso páginas HTML mais longas. Além disso, todos eles se adaptam somente ao espaço disponível no ecrã. No entanto, é possível observar-se um outro padrão que segue uma abordagem adaptativa bastante diferente das que vimos até agora, ao servir-se do espaço que existe fora do ecrã para colocar algum conteúdo ou mesmo a navegação do *site* (ver Figura 30).

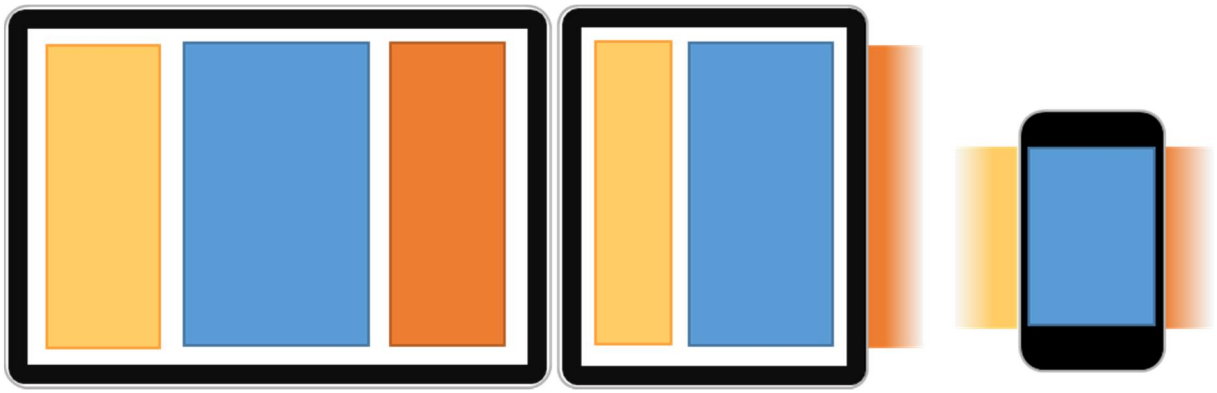


Figura 30. Adaptação de um *layout* com barras laterais (adaptação *off-canvas*)

Este padrão tira vantagens do espaço fora do ecrã (*off-canvas*) para esconder conteúdo até que o ecrã seja suficientemente grande para expor esses componentes. Sob o ponto de vista técnico, o padrão pode ser criado, por exemplo, definindo margens negativas como ponto de partida, e o acesso ao conteúdo escondido é conseguido através de alterações no valor inicial das margens por JavaScript e transições de CSS3. Este padrão evita que os utilizadores naveguem em páginas demasiado longas nos dispositivos mais pequenos. Por outro lado, a sua implementação é mais complexa e implica o recurso a JavaScript.

3.1.2 Tabelas

A tabela é um elemento com uma estrutura pouco flexível. Quando tentamos redimensionar uma tabela em largura, rapidamente atingimos o ponto em que o tamanho da célula não é suficiente para suportar o seu conteúdo. Existe portanto a necessidade de encontrar soluções que resolvam esta questão da adaptação de tabelas.

Para que seja possível visualizar uma tabela num dispositivo com ecrã pequeno é necessário recorrer a algumas transformações, como por exemplo priorizar colunas (em que são escondidas as menos importantes), e ainda adaptar o conteúdo da tabela a uma lista ou mesmo usar os dados da tabela para construir um gráfico. Estas são três possíveis abordagens que dão origem as três padrões de adaptação distintos para tabelas. É ainda possível utilizar um padrão semelhante ao *off-canvas*, apresentando uma parte da tabela enquanto a restante permanece fora do ecrã.

No primeiro padrão identificado, à medida que o tamanho de ecrã disponível vai diminuindo, são exibidas sucessivamente menos colunas. No entanto, este comportamento por si só estaria a limitar o acesso à informação por parte dos utilizadores. Se for dada a possibilidade ao utilizador de escolher quais as colunas visíveis em cada momento (utilizando JavaScript), eliminamos a limitação apontada – ver Figura 31.

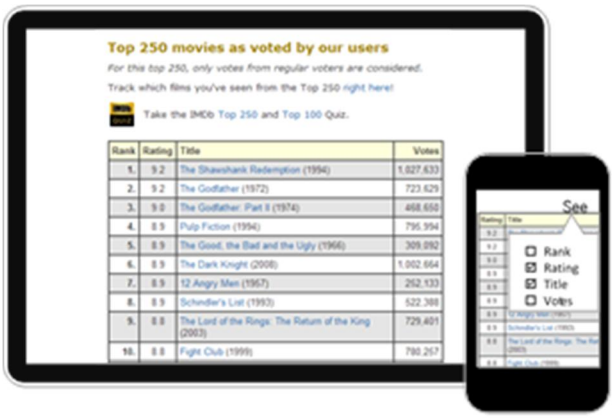


Figura 31. Exemplo da adaptação de uma tabela, omitindo as colunas menos importantes

No segundo padrão, a informação da tabela é convertida numa lista – ver um exemplo da Figura 32. Para isso, é necessário alterar o comportamento natural da tabela, em que cada célula se comporta como uma linha quando são detetados dispositivos pequenos. Cada linha é etiquetada com o nome da coluna a que corresponde. Esta transformação é feita utilizando apenas técnicas de CSS. No entanto, como pontos negativos podemos apontar o facto de tornar as páginas demasiado longas quando lidamos com tabelas com muitos dados. Para além disso, como a estrutura da tabela é completamente transformada, a comparação entre diferentes células torna-se uma tarefa bastante difícil.

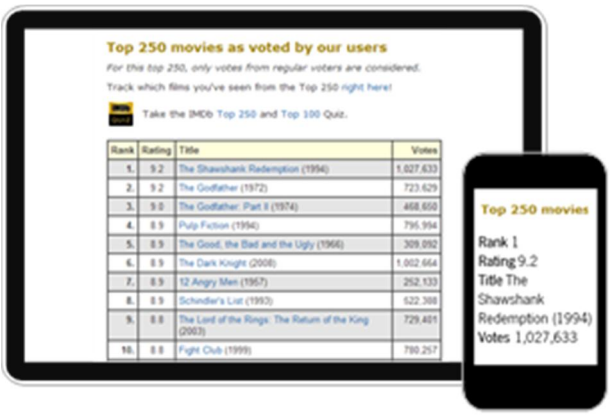


Figura 32. Exemplo da adaptação de uma tabela numa lista

Por sua vez, a transformação da tabela num gráfico pode, em algumas situações, resumir a informação tornando-a mais simples de interpretar. Um exemplo desta transformação pode ser visto na Figura 33. Esta solução pode ser feita com o auxílio de bibliotecas JavaScript, como por exemplo jqPlot. No entanto, este padrão de adaptação não é aplicável em todas as situações, servindo apenas tipos particulares de tabelas que podem ser convertidos em gráficos.

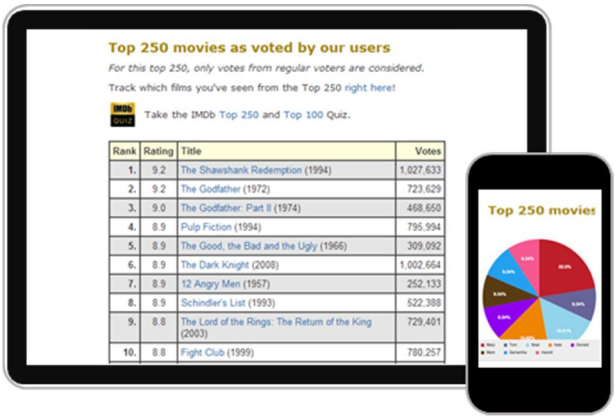


Figura 33. Exemplo da adaptação de uma tabela num gráfico

Por fim, outro padrão observado consiste em manter fixa a primeira coluna (que deve conter o cabeçalho da tabela), e as restantes podem ser percorridas horizontalmente – ver exemplo da Figura 34. Com esta solução é eliminado o excessivo *scroll* vertical e permite-se, ainda, a comparação linha-a-linha.

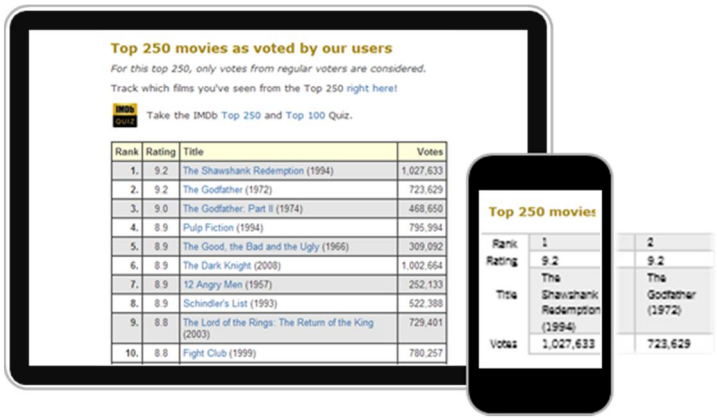


Figura 34. Exemplo da adaptação de uma tabela, utilizando *scroll* horizontal

3.1.3 Navegação

Um componente que merece especial atenção no contexto dos dispositivos móveis é a navegação. Uma das características mais convenientes é que ocupe o menor espaço possível no ecrã. Para além de estar devidamente enquadrado no *site*, o menu deve ser percetível e intuitivo, querendo isto dizer que o utilizador deve rapidamente perceber onde é que se encontra o menu e como utilizá-lo. Nesta secção são ilustrados os padrões estudados para obter uma navegação adaptativa, adequada a dispositivos de diferentes tamanhos.

A primeira solução, e também a mais simples, é manter a navegação fluida, com os itens no topo, independentemente do tamanho do dispositivo, como se ilustra na Figura 35.

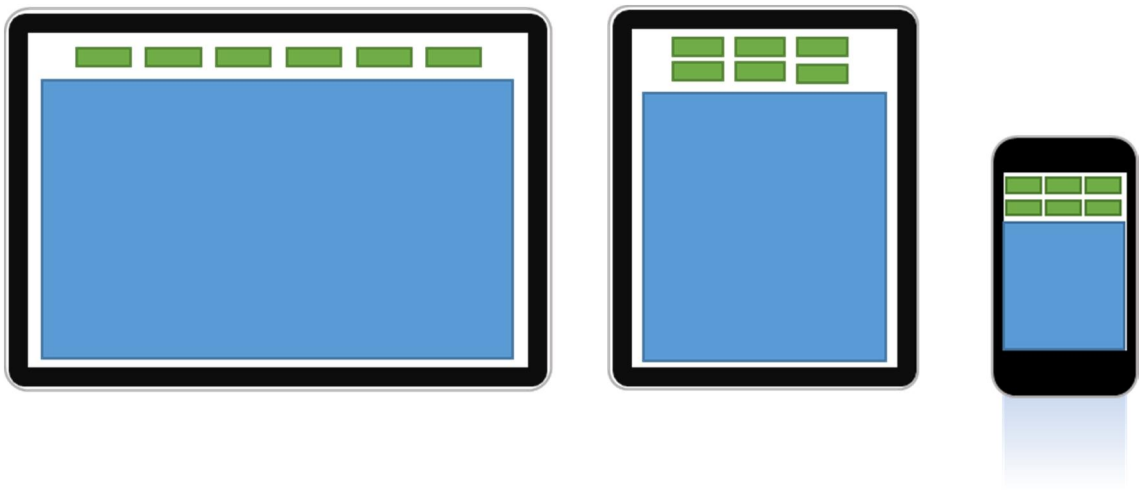


Figura 35. Adaptação da navegação ao tamanho do ecrã

Esta pode ser uma opção para *sites* com um número reduzido de itens de menu. Por outro lado, este padrão de adaptação não é de todo apropriado para aplicações com um grande número de itens de menu. Além disso, é impraticável quando existem opções de submenu.

A principal vantagem está na simplicidade de implementação (não é necessário adicionar HTML extra) e independência de JavaScript. Em contrapartida, este tipo de navegação tem tendência para ocupar demasiado espaço em altura, o que vai contra a ideia de que o conteúdo deve surgir primeiro do que a navegação (*content-first, nav-second*) defendido por Wroblewski (2011). Outros problemas associados a esta navegação são o facto de não ser escalável, e de poder tornar-se difícil de utilizar (no caso de a disposição dos itens ser próxima demais).

Um outro padrão que pode ser aplicado é o *footer anchor*, que consiste em manter a navegação no rodapé (*footer*) do *site*. Existe um *link* que serve com âncora para a navegação do *site*, localizado no topo, e que permite aceder rapidamente ao menu (ver Figura 36).

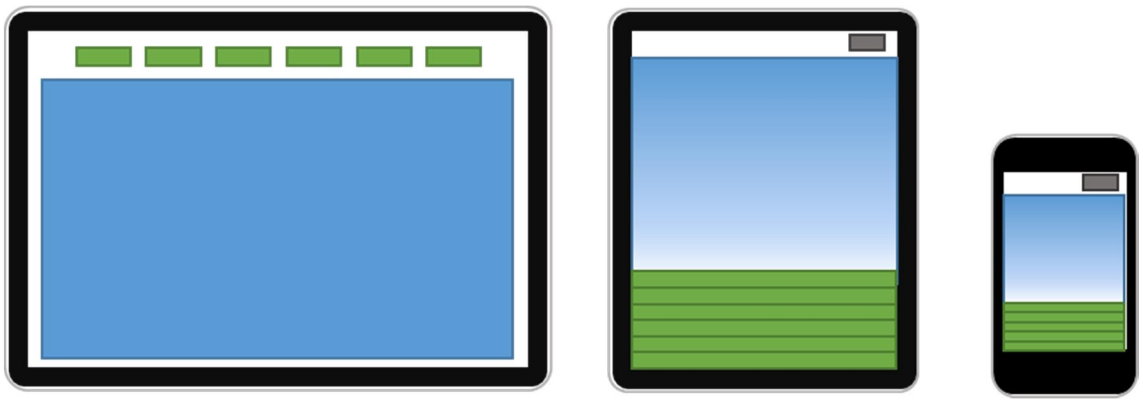


Figura 36. *Footer anchor*

Esta é uma solução simples, e que vai de encontro à filosofia *content-first, nav-second*. O facto do menu se localizar no rodapé da página pode, no entanto, ser um fator negativo. A rápida transição para o rodapé pode suscitar desorientação junto dos utilizadores, isto porque são levados do cabeçalho para o rodapé do *site*.

Um padrão um pouco mais comum é transformar a navegação num componente de seleção (*select* ou *drop down*), vulgarmente utilizado em formulários, para dispositivos com menor resolução (ver Figura 37). A interação com a navegação é mantida no topo da página, e consegue ser facilmente reconhecida com um título "Menu" ou "Navegação". Cada *browser* exibe o componente *select* com o seu próprio aspeto.

Figura 37. *Select* menu

Esta solução é adequada para as situações cuja navegação contém muitos itens ou ainda quando existem vários níveis de navegação. Os subitens são assinalados através de indentação. Qualquer que seja o número de itens do menu, o tamanho do *select menu* ocupará sempre a mesma área, eliminando assim todos os problemas relacionados com o espaço ocupado. A sua implementação requere a utilização de JavaScript para transformar a navegação num componente de seleção.

Um outro padrão bastante comum é o *toggle*. Este pode ser identificado através de um ícone de menu que quando clicado expande e retrai a navegação do *site* (esta ação é conhecida por *toggle* – ver Figura 38). Nesta abordagem não é necessário alterar o elemento que suporta o menu, ao contrário do padrão anterior, que utiliza o elemento *select* em dispositivos com ecrãs

menores. Alguns exemplos de *sites* conhecidos que utilizam esta solução são a Microsoft[23], Nokia[24], Smashing Magazine[25] e Starbucks[26].

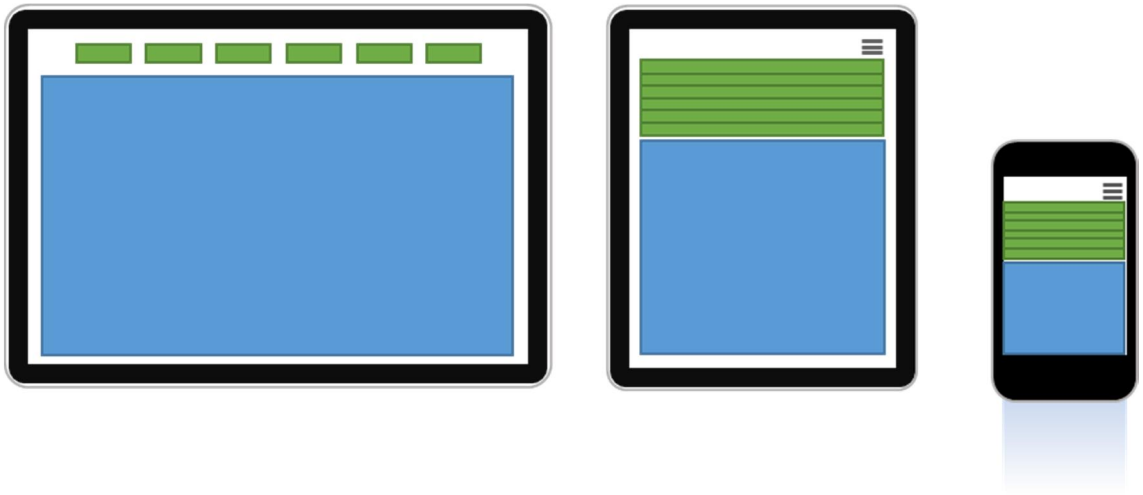


Figura 38. *Toggle* menu

Com esta técnica, o menu mantém-se no topo, e pode ser rapidamente aberto ou fechado, sendo este controlo feito por JavaScript. Esta é também uma solução escalável, isto é, é possível adicionar facilmente novos itens, e pode suportar mais do que um nível de navegação. Para menus mais complexos, com vários níveis, uma solução adequada será usar uma aproximação *multi-toggle*, ou seja, cada item é expansível, e contem a navegação do nível seguinte. Assim, cada nível pode ser expandido para mostrar os respetivos subitens, e retraído para os esconder, fazendo um efeito mola.

Por fim, o padrão *off-canvas* apresentado anteriormente pode também ser aplicado especificamente à navegação de um *site*. Consiste em utilizar uma coluna lateral flutuante, geralmente colocada à esquerda, onde se encontra a navegação (ver Figura 39). O acesso ao menu é feito através de um ícone, tal como o padrão *toggle*. No entanto, neste caso a navegação surge à esquerda (à direita também é possível) e acompanha a altura do *site*. Esta abordagem

[23] http://www.microsoft.com/pt-pt/default.aspx (acedido em 18/10/2013)

[24] http://www.nokia.com/global/(acedido em 18/10/2013)

[25] http://www.smashingmagazine.com (acedido em 18/10/2013)

[26] http://www.starbucks.pt/ (acedido em 18/10/2013)

foi introduzida pelo Facebook, e é bastante comum em aplicações nativas de plataformas móveis.

Figura 39. *Off-canvas* menu

Com esta solução é tirado proveito do espaço fora do ecrã e, ao mesmo tempo, oferece uma experiência de utilização semelhante às aplicações móveis. Uma votação recente que questionava os utilizadores sobre se este padrão de navegação é apropriado tanto para dispositivos móveis como *desktop* deu origem a um resultado favorável a esta implementação: apenas 10% dos utilizadores votaram desfavoravelmente, alegando que este padrão não é bom para nenhum *site*. 46% dos utilizadores votaram dizendo que este é um padrão bom para qualquer site e 44% disseram que é um bom padrão apenas para *sites* quando visualizados em dispositivos móveis (CSSTricks, 2013). O Feedly[27] é um exemplo de uma aplicação que utiliza este padrão de navegação independentemente do tamanho do ecrã do utilizador.

Esconder a navegação pode ainda ser uma solução, ainda que pouco adequada na maioria dos casos. Embora liberte espaço, o que é importante nos dispositivos com ecrãs pequenos, esconder informação e tornar inacessíveis páginas de conteúdo penaliza os utilizadores. No caso de a navegação estar ausente, é fundamental existir uma área de pesquisa, para que o utilizador consiga alcançar o conteúdo que procura.

[27] http://cloud.feedly.com (acedido em 19/10/2013)

3.2 Otimização

Até agora vimos diferentes padrões de adaptação que resolvem um dos maiores desafios do *responsive web design*, o de conseguirmos adaptar o conteúdo ao tamanho do dispositivo. No entanto, a diferença entre os diversos dispositivos não se limita ao tamanho do ecrã. Por exemplo, a forma de interação com um *smartphone* é maioritariamente feita através do toque. Além disso, não podemos descurar que o propósito essencial de um telemóvel é permitir fazer chamadas. De forma a tirar partido de algumas dessas características e capacidades dos dispositivos podemos fazer pequenas otimizações com o objetivo de oferecer ao utilizador uma experiência mais adequada a cada dispositivo.

3.2.1 Otimização ao toque

Uma otimização que pode ser feita é utilizar o tipo de *input* de HTML5 apropriado. Conforme seja um campo de *email*, *url* ou numérico, podemos apresentar o teclado apropriado. Por exemplo, se em vez de utilizarmos o tipo mais comum `text` utilizarmos o tipo `email`, nos dispositivos iOS o teclado possui o caracter @ para tornar mais fácil o seu acesso, como é visível na Figura 40.

```
<input type="text" name="email" id="email" />
<input type="email" name="email" id="email" />
```

Figura 40. Diferença entre os *inputs* de HTML5 *text* e *email*

Podemos ver a diferença no teclado com os tipos de *input url* e numérico na Figura 41. Embora sejam pequenas otimizações, conseguimos favorecer a experiência de utilização.

Figura 41. *Inputs* de HTML5 *url* e *number*

Outra otimização possível é tirar partido da capacidade de fazer chamadas dos telemóveis. Alguns dispositivos reconhecem um número de telefone e permitem imediatamente iniciar uma chamada. Para isso basta definir um número de telefone da seguinte forma:

```
<a href="tel:+808200300">808 200 300</a>
```

A sensibilidade ao toque de alguns dispositivos deve jogar a favor do utilizador, permitindo-lhe interagir com o *site* de uma forma otimizada ao tipo de dispositivo que utiliza. Por exemplo, basta tocar com dois dedos num artigo do jornal *The Boston Globe* para que poder guarda-lo para ler mais tarde e *off-line* (Figura 42).

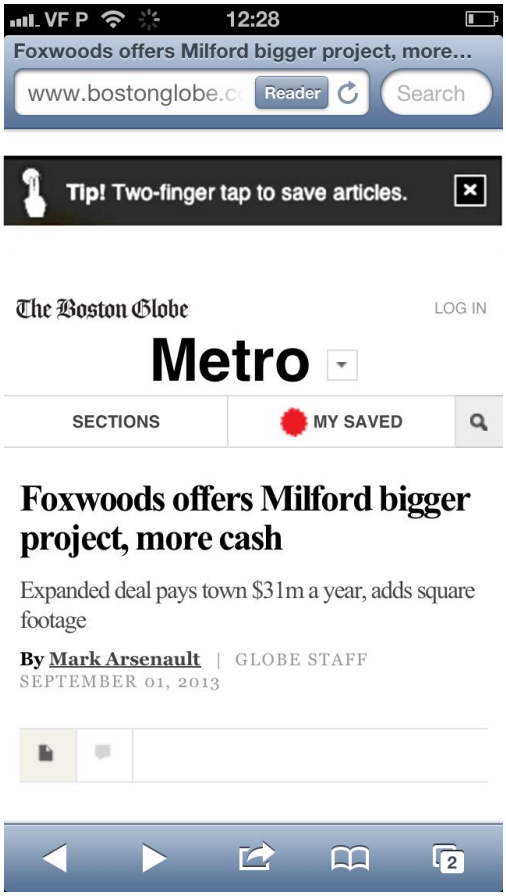


Figura 42. Exemplo de uma otimização para dispositivos sensíveis ao toque

3.2.2 Desempenho

O desempenho é um fator importante para o sucesso de um *site*. Segundo Frost (2012), 71% dos utilizadores esperam que um *website* carregue pelo menos tão rápido em dispositivos móveis como em computadores, e 74% abandonam o *site* se ele demorar mais de 5 segundos a ser carregado. Além disso, 86% dos *websites responsive* apresentam o mesmo tamanho das páginas tanto em pequenos como em grandes ecrãs, e atualmente uma página *web* tem em média 1MB. No entanto, sabemos que os dispositivos móveis têm características muitos diferentes dos *desktops*. A mobilidade destes dispositivos faz com que sejam utilizados em qualquer lugar, sobre condições e contextos difíceis de prever. Por exemplo, segundo um estudo levado a cabo pela (Bulger 2010), 76% dos utilizadores utilizam o seu telemóvel enquanto aguardam por uma reunião ou encontro, e 69% utilizam enquanto fazem compras.

- Limitações na conexão à Internet

Existe uma grande diferença entre uma conexão *wireless* de alta velocidade e uma rede de um telemóvel. Mas o tipo de conexão é apenas um começo: há outros fatores influenciadores como a localização, a operadora ou o número de pessoas ligadas à mesma rede. Por isso, não é correto fazer assunções sobre o desempenho de uma conexão com base no dispositivo utilizado, nem no tipo de conexão existente, já que uma ligação 3G pode ser mais rápida do que uma conexão lenta de Wi-Fi. Conhecer a largura de banda é uma técnica mais eficiente para determinar as condições reais da ligação à internet.

Ter a informação mais precisa sobre a qualidade da conectividade do dipositivo utilizado permite ao *developer* tomar decisões, por exemplo, sobre qual a resolução mais apropriada para as imagens.

- Utilização de recursos desnecessários

Por vezes é assumido que suportar todos os dispositivos significa concentrar, independentemente do contexto atual, todos os recursos necessários. A utilização de certos recursos como por exemplo *scripts*, imagens e vídeos podem contribuir com fortes atrasos no carregamento de uma página *web*. Evitar que certos recursos sejam carregados quando não são usados em determinados contextos é uma forma de melhorarmos o desempenho, ao diminuir a latência no acesso às páginas.

Carregar recursos de forma condicional (*conditional loading*) consiste em inspecionar as condições do utilizador, como por exemplo o tamanho do dispositivo, e em função dessa informação carregar o conteúdo apropriado. Por exemplo, se for detetado que o utilizador tem um dispositivo com ecrã pequeno, é possível não carregar um vídeo (e, por exemplo, substituir por um *link* para o vídeo), evitando assim o desperdício de largura de banda e, dependendo da qualidade da conexão do utilizador, diminuindo o tempo de carregamento da página. E se em vez de apenas um carregarmos condicionalmente vários recursos, é possível diminuir significativamente o tempo de espera.

Outra forma de se conseguir atingir tempos de espera menores no carregamento de um *site* é optarmos por utilizar uma estratégia em que só são carregamos os recursos quando são necessários. Esta técnica é designada *lazy loading* e pode ser implementada em JavaScript. Um exemplo onde esta estratégia pode ser aplicada é no carregamento da secção de comentários de

uma página, onde os comentários são carregados à medida que o utilizador faz *scroll* nessa área.

O conjunto de otimizações explorado nesta secção ilustram algumas estratégias que podem ser adotadas com o objetivo de diminuir o tempo de acesso à informação, considerando as condições e o contexto em que o utilizador se encontra. Descarregarmos apenas os recursos necessários em cada contexto traz também outras vantagens associadas como diminuir a largura de banda e a memória gasta, e ainda poupar bateria, que nos dispositivos móveis tem um papel muito importante.

3.3 Conclusão

Ao longo deste capítulo foram descritas diferentes soluções padrão para resolver as questões relativas às necessidades adaptativas de interfaces *web*, conforme o tamanho do ecrã do dispositivo. O objetivo principal é manter a informação acessível seja qual forma o tipo de dispositivo utilizado, criando assim condições para suportar uma alargada gama de dispositivos com diferentes tamanhos.

Com a adaptação conseguimos garantir que uma dada aplicação *web* se manterá utilizável independentemente do tamanho do ecrã, com o conteúdo sempre acessível em qualquer momento. No entanto, para melhorar a experiência de utilização podem ser feitas otimizações conforme determinados indicadores do *browser*, aproximando, assim, a experiência de utilização ao contexto do utilizador. É desta forma que conseguimos oferecer uma experiência adaptada à plataforma em que esta ocorre. Aqui reside a diferença entre otimizar para um determinado conjunto de dispositivos ou apenas suportá-los.

O próximo capítulo é dedicado à implementação dos diferentes padrões de adaptação de menus ilustrados na Secção 3.1.3. Para além de garantir o suporte de uma navegação adaptada e funcional, dependendo do tamanho do dispositivo utilizado, são ainda feitas algumas otimizações conforme a capacidade do *browser* e do dispositivo, com o objetivo oferecer uma experiência de utilização mais adequada e adaptada em cada caso.

Capítulo 4

Adaptação automática de menus para responsive web designs

Uma ideia subjacente ao capítulo anterior é a de que mesmo que não seja possível otimizar para todos os dispositivos, devemos tentar suportá-los ao tornar o seu conteúdo sempre acessível. A navegação adaptada ao tamanho do dispositivo utilizado serve exatamente para que os utilizadores possam navegar em qualquer *site* e alcançar o conteúdo que procuram. Implementar uma solução que coincide com um dos padrões de navegação apresentados no capítulo anterior é, por isso, uma ação fundamental no desenvolvimento de um *site responsive*.

Um dos objetivos propostos consiste em desenvolver uma solução que integre os diferentes padrões de adaptação de menus estudados, e que seja capaz de tornar automático o processo de transição de um menu para se obter uma navegação adaptada aos dispositivos com ecrã reduzido. A implementação desta solução vai permitir às aplicações ou *sites* a adaptação da sua navegação ao tamanho de dispositivo de uma forma automática. Deste modo, a navegação em dispositivos móveis deixa de ser uma preocupação para o *designer* (ou *developer*), economizando esse tempo de desenvolvimento. Ao suportar os diferentes padrões, é ainda possível que seja escolhida a abordagem adaptativa que mais se adequa a cada caso específico.

4.1 Deteção de características do dispositivo

4.1.1 Deteção do lado do servidor

A deteção do lado do servidor consiste em explorar a *string user agent* enviada pelo *browser* quando uma página *web* é requisitada. O seu propósito é determinar o cliente (*browser*) que está a ser utilizado, e a partir dai é possível obter informações detalhadas sobre as suas características suportadas, e ainda sobre as capacidades do dispositivo. A deteção a partir de *user agents* engloba um conjunto amplo de informações, como verificar se o *browser* suporta

gradientes de CSS, e se o dispositivo ao qual esse *browser* pertence pode ou não fazer chamadas.

Segue-se um exemplo de uma *string user agent*:

> Mozilla/5.0 (Linux; U; Android 2.2.1; en-us; SCH-R880 Build/FROYO)
>
> AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1

Daqui podemos retirar algumas informações pertinentes. Por exemplo as seguintes:

- Android 2.2.1 – trata-se de um dispositivo com sistema operativo Android, na versão 2.2.1;
- AppleWebKit/533.1 – diz respeito ao motor de *layout* (responsável pela renderização do página *web*) e o número de série;
- SCH-R880 – dá-nos a referência de que o dispositivo é da marca Samsung.

Algumas das *string user agents* mais comuns são identificadas na Tabela 4.

Tabela 4. Descrição dos *user agents* mais comuns em 2013[28]

User agent	Browser/Sistema
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:23.0) Gecko/20100101 Firefox/23.0	Firefox 23.0 Win7 64-bit
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) AppleWebKit/536.30.1 (KHTML, like Gecko) Version/6.0.5 Safari/536.30.1	Safari 6.0 MacOSX
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.66 Safari/537.36	Chrome 29.0 Win7 64-bit
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:23.0) Gecko/20100101 Firefox/23.0	Firefox 23.0 MacOSX
Mozilla/5.0 (iPhone; CPU iPhone OS 6_1_3 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10B329 Safari/8536.25	Mobile Safari 6.1 iOS

[28] http://techblog.willshouse.com/2012/01/03/most-common-user-agents/ (acedido em 20/09/2013)

Os repositórios de deteção de dispositivos (DDR) como o WURFL e o DeviceAtlas, constituem listas com informações detalhadas sobre dispositivos, *browsers* e sistemas operativos. É a partir da consulta desses repositórios que conseguimos identificar as características do *browser* utilizado. Utilizando por exemplo o WURFL, podemos obter um resultado de cerca de 500 características diferentes a partir da deteção e interpretação de *user agents*. Com a informação devolvida é possível oferecer diferentes experiências de utilização conforme o *browser* e o dispositivo utilizado, providenciando o HTML, CSS e JavaScript necessários para suportar um determinado dispositivo.

Os pontos fortes da deteção com *user agents* é o facto de se conseguir obter uma informação muito pormenorizada sobre os dispositivos utilizados. Além disso, como a deteção é feita do lado do servidor, evita o consumo de um recurso extra do lado do cliente. Por outro lado, os resultados obtidos podem ser pouco precisos (podem ser manipulados pelos *browsers*), e para se obter um grande conjunto de informação é geralmente necessário recorrer a um serviço externo, o que é uma despesa extra para o projeto.

4.1.2 Deteção do lado do cliente

A deteção do lado do cliente pode ser feita utilizando bibliotecas de JavaScript para testar se uma determinada característica ou funcionalidade é suportada pelo *browser*. Conforme exista ou não, podemos determinar o comportamento da página HTML ou de um ou vários elementos.

Modernizr é uma biblioteca de JavaScript que ajuda na deteção das funcionalidades suportadas pelo *browser*. Inclui mais de 40 testes diferentes, entre os quais verificam o suporte de HTML5, CSS3, se o *browser* suporta JavaScript, *media queries*, se dispositivo é sensível ao toque e possui uma API de geolocalização. A sua estrutura modular permite utilizar apenas algumas opções de teste, evitando assim carregar recursos que não são necessários. Conforme o resultado dos testes, o Modernizr adiciona ao elemento HTML classes que indicam as funcionalidades suportadas e não suportas.

Um teste utilizando as características por omissão da ferramenta, realizado no *browser* Chrome (versão 26), resultou na adição das classes seguintes:

```
<html class="js flexbox canvas canvastext webgl no-touch
geolocation postmessage websqldatabase indexeddb hashchange
history draganddrop websockets rgba hsla multiplebgs
backgroundsize borderimage borderradius boxshadow textshadow
opacity cssanimations csscolumns cssgradients cssreflections
csstransforms csstransforms3d csstransitions fontface
generatedcontent video audio localstorage sessionstorage
webworkers applicationcache svg inlinesvg smil svgclippaths">
```

O mesmo teste utilizando o Internet Explorer (versão 9), obteve-se o seguinte resultado:

```
<html class="js no-flexbox canvas canvastext no-webgl no-touch
geolocation postmessage no-websqldatabase no-indexeddb
hashchange no-history draganddrop no-websockets rgba hsla
multiplebgs backgroundsize no-borderimage borderradius boxshadow
no-textshadow opacity no-cssanimations no-csscolumns no-
cssgradients no-cssreflections csstransforms no-csstransforms3d
no-csstransitions fontface generatedcontent video audio
localstorage sessionstorage no-webworkers no-applicationcache
svg inlinesvg smil svgclippaths">
```

Quando uma determinada funcionalidade não é suportada, é atribuído o prefixo "no-". Dos resultados acima, conclui-se então, por exemplo, que o Chrome suporta webgl, enquanto o Internet Explorer não, que nenhum dos *browsers* está a executar num dispositivo com capacidades de interação via toque e que os dois suportam vídeo e áudio. Estas classes podem ser usadas na definição de regras de formatação dos elementos por CSS, contornando as limitações do *browser* e facilitando o *progressive enhancement*. No exemplo seguinte, retirado de (Ates 2010), prende-se adicionar ao elemento com o identificador *content* um contorno redondo e uma sombra à volta do elemento. Com o Modernizr é possível aplicar as formatações adequadas conforme essas características sejam ou não suportadas pelo *browser*. O excerto de código seguinte pretende demonstrar como é possível alcançar a formatação desejada[29]:

[29] Está a omitir-se o prefixo dos diferentes *browsers*.

```
.borderradius #content {
      border: 1px solid #141414;
      border-radius: 12px;
}
.boxshadow #content {
      border: none;
      box-shadow: rgba(0,0,0, .5) 3px 3px 6px;
}
```

Para além das classes que são adicionadas, também é possível aceder ao resultado dos testes através de um objeto Modernizr (utilizando JavaScript), por exemplo, da seguinte forma: `Modernizr.geolocation`. Esta abordagem facilita o carregamento condicional de certos recursos consoante as características suportadas pelo *browser* ou dispositivo.

Embora esta alternativa de deteção tenha menos opções de teste do que a deteção por *user agents*, providencia um conjunto bastante interessante de testes, mais do que suficiente para muitos projetos. É também bastante mais simples de implementar do que utilizando a deteção do lado do servidor, uma vez que não é necessário tratar manualmente a *string user agent* e só a partir daí concluir sobre as características do *browser* utilizado.

A alternativa de deteção realizada do lado do cliente tem, no entanto, algumas limitações. Uma dela é ser dependente de JavaScript para funcionar. Se o JavaScript não for suportado, é oferecido um nível mais baixo de experiência. Além disso, como se trata de uma abordagem do lado do cliente, podem ocorrer problemas de desempenho. No entanto, se forem utilizados apenas os módulos de teste necessários em cada projeto, esse problema pode ser reduzido.

Combinar *responsive web design* com a deteção das capacidades do *browser* permite suportar uma gama mais alargada de dispositivos e, simultaneamente, oferecer experiências de utilização cada vez mais refinadas e otimizadas de acordo com as capacidades e características do dispositivo. Foi decidido que na implementação dos padrões de navegação será utilizado o Modernizr para testar o suporte de JavaScript, de CSS3 e ainda se o dispositivo é ou não recetivo ao toque. Esta escolha deve-se ao conjunto de testes ser suficiente, ao facto de promover o *progressive enhancement*, e ainda à simplicidade e rapidez de implementação, sem quaisquer

custos de desenvolvimento adicionais. Serão utilizados apenas os módulos de teste necessários para otimizar a solução final o mais possível.

4.2 Implementação de padrões de navegação

Ao longo desta secção é descrita a forma como foram desenvolvidos os vários padrões de navegação adaptáveis apresentados anteriormente, explicando a implementação de cada um dos deles: manter a navegação original, converter num *select* menu, utilizar o menu no rodapé, transformar num menu *toggle* e *off-canvas*. Todos os padrões devem refletir um equilíbrio entre a acessibilidade e o espaço que ocupa a navegação.

O ponto de partida para implementar este conjunto de diferentes padrões consistiu num *site* exemplo com uma navegação convencional (formada por uma lista com *links* para outras páginas – ver Figura 44), com apenas alguns estilos de formatação simples. A navegação consiste na barra preta no topo da página (Figura 43). À medida que o *browser* é redimensionado, observamos a sua adaptação ao tamanho do ecrã. Esta adaptação provém do comportamento natural destes elementos e consiste no padrão adaptativo mais simples estudado, mantendo o menu tal como é, qualquer que seja o tamanho do dispositivo. No entanto, a proximidade entre os elementos parece desfavorecer a sua utilização.

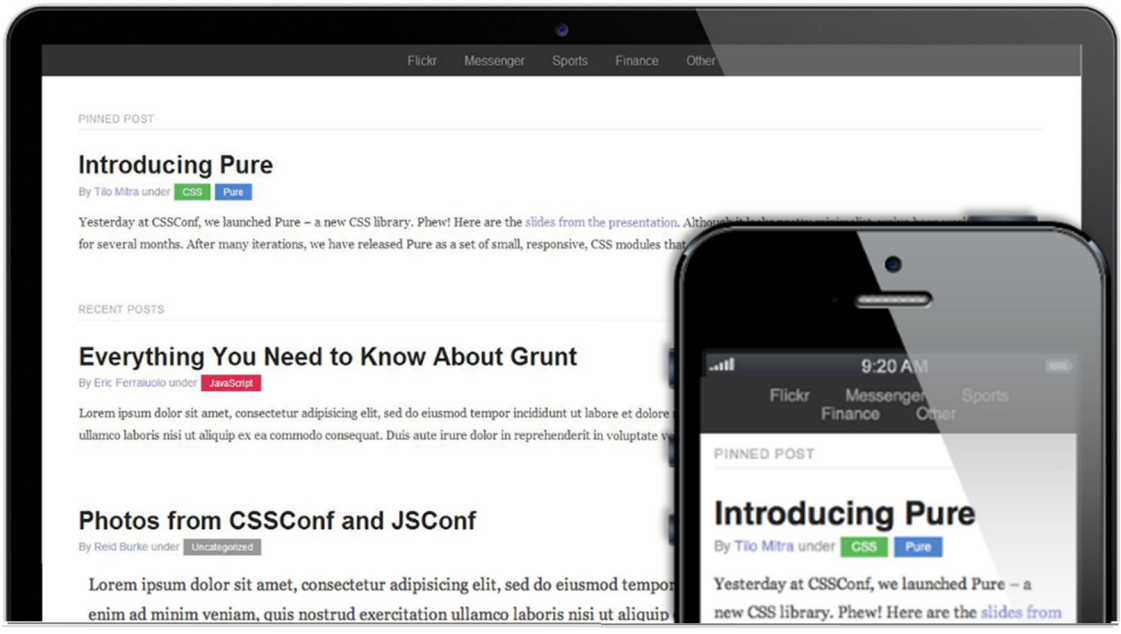


Figura 43. *Site* exemplo para a adaptação de menus

Os padrões seguintes seguem abordagens um pouco mais complexas que envolvem alterações ao nível da DOM e na formatação dos elementos. Para a sua implementação, recorreu-se essencialmente à biblioteca de JavaScript jQuery, com a qual é possível transformar a navegação e controlar todos seus eventos e animações.

Na implementação destes padrões, não foram aplicadas quaisquer formatações *inline* diretamente sobre os elementos. Em vez disso, optou-se por utilizar classes para gerir estados, e assim controlar as formatações necessárias de todos os elementos.

4.2.1 Select menu

O primeiro passo para a transformação da navegação num *select* menu consiste em identificar o elemento da navegação. De seguida, é adicionado à DOM um novo elemento *select*. Cada item do menu é percorrido iterativamente, e adicionado ao *select* menu. O antes e depois é ilustrado nas Figuras 44 e 45.

```
▼<ul>
  ▼<li>
     <a href="/flickerandme">Flickr & Me</a>
   </li>
  ▼<li>
     <a href="/messenger">Messenger</a>
   </li>
  ▼<li>
     <a href="/sports">Sports</a>
   </li>
  ▼<li>
     <a href="/finance">Finance</a>
   </li>
  ▼<li>
     <a href="/other">Other</a>
   </li>
 </ul>
```

Figura 44. Estrutura original da navegação

```
▼<select>
    <option selected="selected" value>Menu</option>
    <option value="/flickerandme">Flickr & Me</option>
    <option value="/messenger">Messenger</option>
    <option value="/sports">Sports</option>
    <option value="/finance">Finance</option>
    <option value="/other">Other</option>
  </select>
```

Figura 45. Estrutura resultante da adaptação da navegação utilizando o *select* menu

O controlo do menu a apresentar em cada momento é feito através de uma *media query*, exibindo a navegação mais adequada conforme o tamanho do ecrã.

```
.js nav select {
    display: none;
}

@media (max-width: @breakpoint) {
    .js nav {
        ul { display: none; }
      }
}
```

Em ecrãs com tamanho superior ao valor do *breakpoint*, a navegação apresentada é a original do *site*. A Figura 46 ilustra o resultado quando o valor do *breakpoint* é inferior à janela de visualização. A adaptação só ocorre se o JavaScript for suportado no *browser*, o que é controlado pela class `js`. Caso contrário, não só o elemento *select* não seria injetado na página, como a navegação original era simplesmente escondida – ou seja, o menu desapareceria em ecrãs mais pequenos.

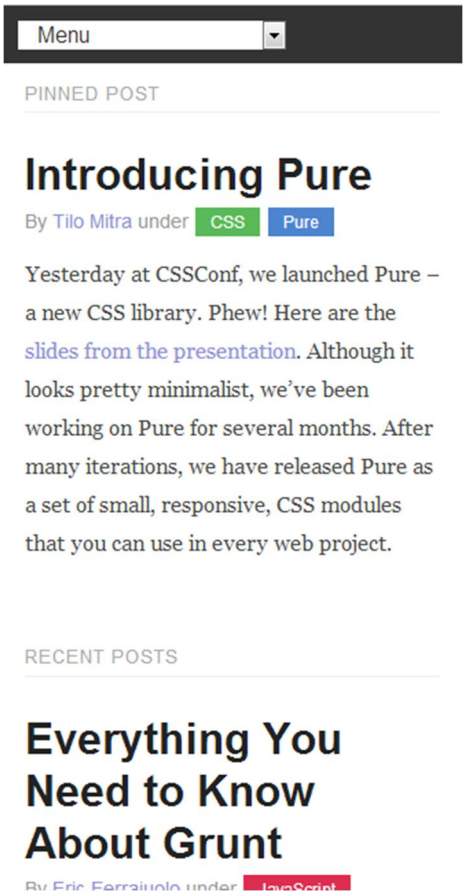


Figura 46. *Select* menu

Nesta implementação o elemento *select* é povoado pelos itens do menu existentes, ao contrário da abordagem seguida por (Pietrusky 2012), que implementa este padrão criando o menu diretamente na página HTML em vez de o construir dinamicamente. Se por um lado a alternativa do autor não necessita de JavaScript, por outro a duplicação de menus dificulta a sua manutenção.

4.2.2 Footer anchor menu

O *footer anchor* consiste em tirar partido do espaço no fundo da página HTML para aí colocar a navegação, utilizando uma âncora para acesso rápido no cabeçalho. Este padrão poderia ser implementado sem recorrer a JavaScript, utilizando *media queries* com diferentes formatações CSS para manipular a forma como é apresentado o menu em diferentes tamanhos da janela de visualização. Por exemplo, em ecrãs pequenos seria apresentado um item por linha:

```
[data-role=”navigation”] li a {
      display: block;
}
```

Já em ecrãs maiores o menu corresponderia a uma única barra de navegação no topo:

```
@media screen and (min-width: @breakpoint) {
      [data-role=”navigation”] {
            position: absolute;
            li {
                  display: inline-block;
            }
      }
}
```

No entanto, isso implicaria que o menu estivesse posicionado no fim da página (o que não acontece na maioria dos *sites*). Como o objetivo desta solução passa também pela adaptação de menus já existentes, foi necessário recorrer a JavaScript para reposicionar a navegação na DOM, seja qual for a sua localização original, uma vez que apenas com CSS isso não seria possível.

Através de jQuery a navegação é posicionada no elemento *footer* (rodapé). É ainda adicionado um *link* “Menu”, elemento que funcionará como âncora para a navegação se tornar facilmente acessível (ver Figura 47).

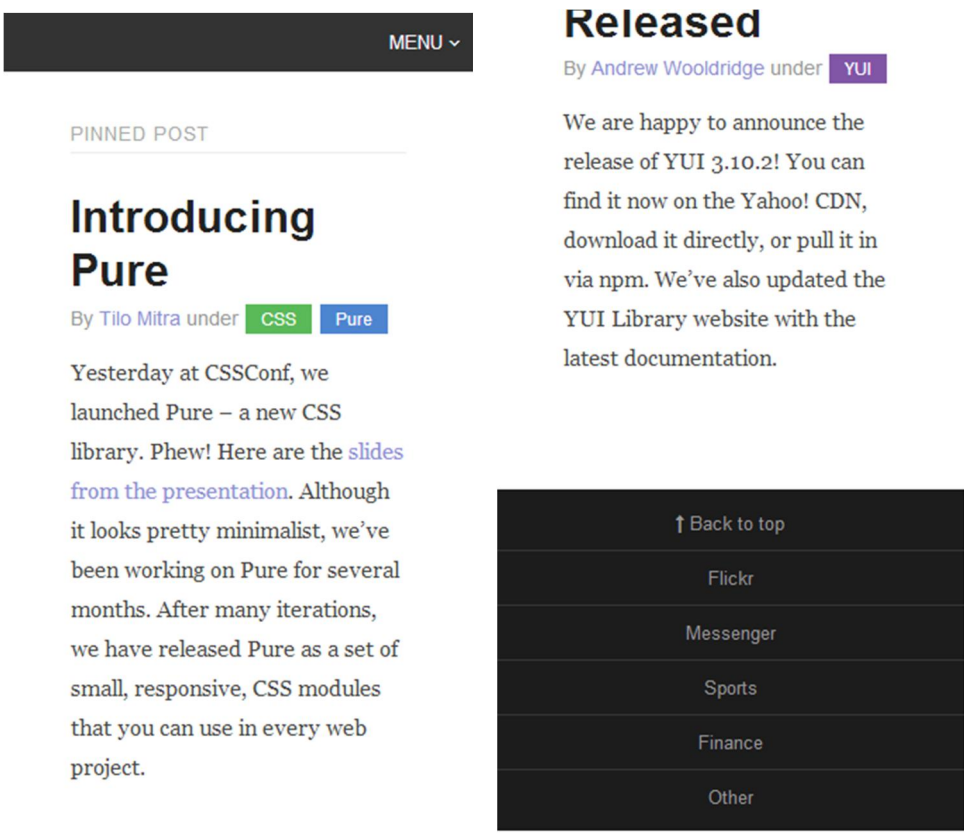


Figura 47. *Footer anchor* menu

Uma das desvantagens que pode ser apontada a este padrão adaptativo é o facto de poder causar desorientação aos utilizadores, que de repente são conduzidos para o fundo da página. Regressar ao topo implica percorrer manualmente toda a página (*scroll* vertical). Para contornar esse inconveniente, foi criada uma opção "voltar ao topo" que rapidamente conduz o utilizador novamente para o ponto onde se encontrava.

4.2.3 Toggle menu

Assim que a página é carregada, é determinada a largura do dispositivo que está a ser usado, utilizando o método jQuery `.innerWidth`. Conforme o valor da largura for maior ou menor do que o valor definido como o valor de transição (*breakpoint*), é apresentado o menu no formato mais adequado. Este padrão segue uma abordagem em que a navegação é reorganizada tendo em conta algumas regras de adaptação aplicadas, para que daqui resultasse um menu 100% funcional em dispositivos com ecrãs mais pequenos. As ações de abertura e

fecho do menu são controladas por JavaScript. Na figura 48 é ilustrado o resultado da adaptação utilizando este padrão.

Detalhando agora as adaptações necessárias, começamos por adicionar uma classe `.toggle` no elemento *body*, criando uma espécie de invólucro (*wrapper*), a partir do qual são especificados algumas regras de CSS para formatar o menu da forma adequada. Este procedimento garante que as formatações do menu adaptado não intervém nas já existentes. É ainda adicionada uma classe `.parent` aos elementos com subitens, para que se possa controlar também a subnavegação.

Posteriormente é criada uma nova estrutura que suportará todo o menu. Essa estrutura é composta pelos itens de menu e por uma âncora (representada pelo ícone de menu), que serve para interagir com o menu. Quando a página é carregada o menu está escondido, e as ações de abertura e fecho são despoletadas pelo clique (ou toque) no ícone de menu.

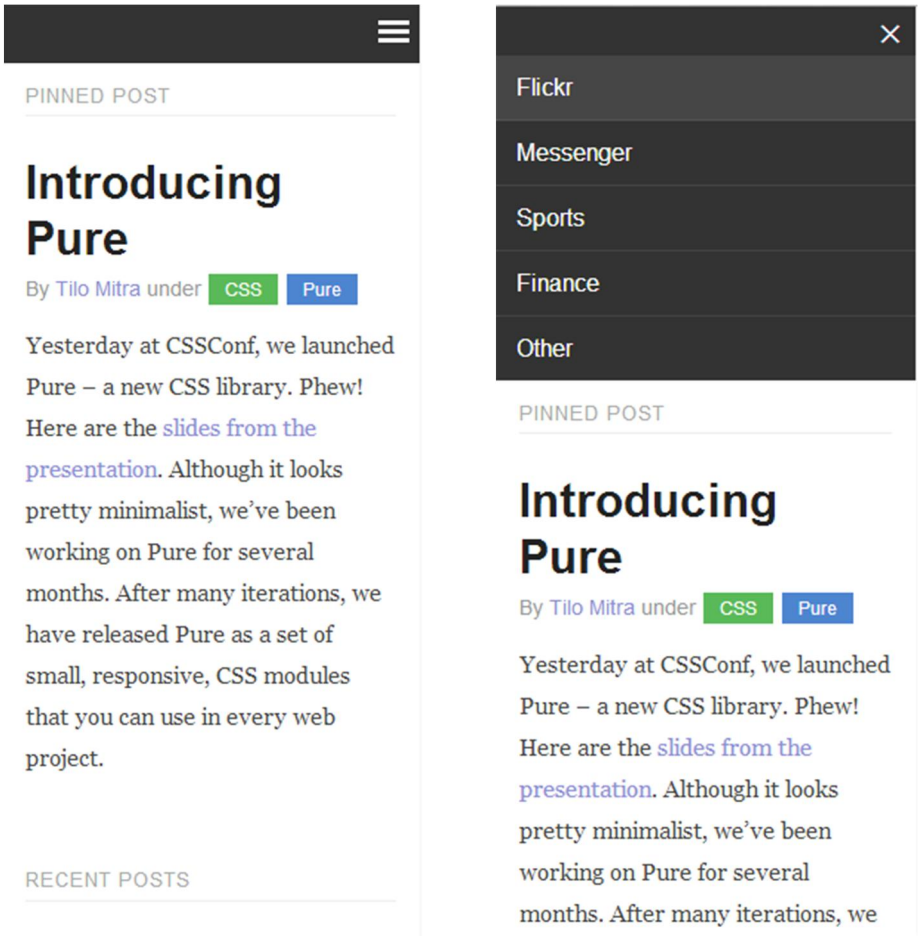


Figura 48. *Toggle* menu

Como é necessário garantir que em cada momento é apresentada a navegação mais adequada, precisamos de conseguir alternar entre a navegação original e adaptada. Desta forma, para restabelecer o menu original do *site*, basta garantir que são removidas as classes adicionadas durante o conjunto de passos anteriormente descritos. Deste modo garantimos que as formatações aplicadas desenham o menu original.

4.2.4 Off-canvas menu

Deste conjunto de padrões de navegação, o *off-canvas* é o padrão mais complexo, uma vez que envolve a alteração de propriedades de outros elementos para além do menu propriamente dito. No entanto, este é o padrão adaptativo mais próximo de uma experiência semelhante às aplicações nativas de plataformas móveis. A seguir são descritos alguns detalhes de implementação mais relevantes.

Primeiramente, foi adicionado um invólucro `.wrapper` que envolve todos os elementos da página HTML presentes no *body*, para permitir a deslocação à esquerda quando o menu é aberto. Foi adicionado o ícone de menu no cabeçalho da página para ser possível aceder à navegação, uma vez que esta permanece fora do ecrã.

Quando o utilizador clica no ícone de menu para o abrir, todo o conteúdo da página HTML é deslocado 70% da esquerda para a direita (Figura 50).

```
.js .wrapper {
      left: 0;
}
.js [data-role=”navigation”] {
      left: -70%;
}
```

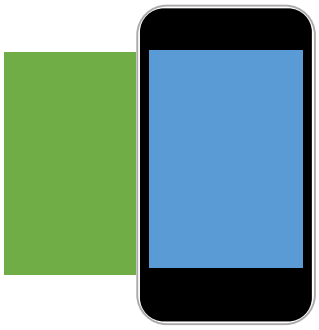


Figura 49. *Off-canvas* (navegação fora do ecrã)

```
.js .wrapper {
      left: 70%;
}
.js [data-role="navigation"] {
      left: 0;
}
```

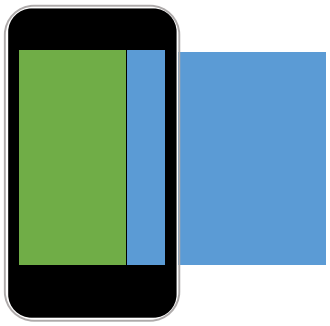


Figura 50. *Off-canvas* (navegação dentro do ecrã)

Ao clicar em qualquer elemento, o menu é fechado e o conteúdo da página volta à posição inicial, descrevendo um movimento da direita para a esquerda (Figura 49).

O efeito deslizante na abertura e fecho do menu foram obtidos, numa primeira abordagem, utilizando o método `.animate` de jQuery. No entanto, ao inspecionar o que acontecia durante a ação de abertura e fecho do menu, reparou-se que o browser recalculava em tempo real todo o layout para determinar a posição do elemento `.wrapper`, para cada *frame* de animação. Como resultado, a posição do elemento estava a ser calculada e injetada diretamente na *markup*, através da técnica de *style inline*. Para além de problemas de desempenho relativos aos cálculos necessários, os estilos *inline* injetados trazem problemas de manutenção que podem ser evitados.

Em busca de uma alternativa ao método `.animate`, chegou-se à conclusão de que o mesmo comportamento poderia ser obtido com melhor desempenho através de transições em CSS3 para criar a animação pretendida. Frost (2012) defende mesmo que devem ser utilizadas transições de CSS3 em vez de jQuery para otimizar questões de desempenho, o que é muito importante, principalmente para otimização em dispositivos móveis.

Utilizando a função de CSS3 `translate` foi possível mover o elemento (assim como o seu conteúdo) da sua posição original para qualquer coordenada x,y e z. Mas como nem todos os *browsers* são capazes de suportar transições em CSS3, foi necessário aplicar a animação com base no conceito de *progressive enhancement* (anteriormente abordado na Secção 2.2.2), assegurando assim a acessibilidade da navegação mesmo nos *browsers* com sérias limitações a

este nível. Neste caso, foi utilizado o Modernizr como ferramenta chave para saber se o *browser* suporta transformações e transições CSS3.

```
.js.csstransforms3d.csstransitions .wrapper {
      left: 0;
      transform: translate3d(0, 0, 0);
      transition: transform 500ms ease;
}
.js [data-role="navigation"] {
      left: 0;
      transform: translate3d(-100%, 0, 0);
}
```

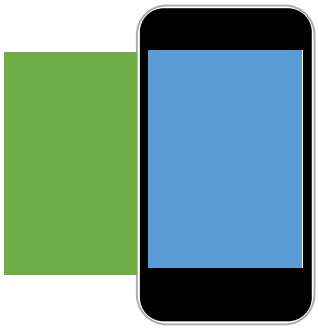


Figura 51. *Off-canvas* com transições de CSS3 (navegação fora do ecrã)

```
.js.csstransforms3d.csstransitions .wrapper {
      transform: translate3d(70%, 0, 0);
      transition: transform 500ms ease;
}
```

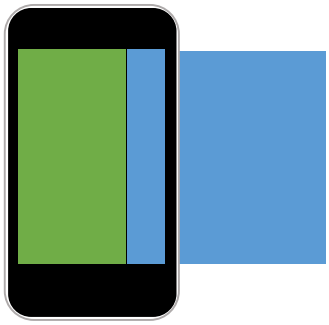


Figura 52. *Off-canvas* com transições de CSS3 (navegação dentro do ecrã)

Como vimos anteriormente na Secção 4.1, o Modernizr é uma biblioteca de JavaScript que inspeciona as características de HTML5 e CSS3 suportadas pelo *browser*. A utilização desta ferramenta permitiu fazer o aumento progressivo das funcionalidades de CSS utilizadas (*progressive enhancement*), conforme estas existam ou não.

Foi ainda adicionada uma classe `.nojs` ao elemento HTML (`<html class="nojs">`). Assim, quando o utilizador tenta aceder à página HTML, se o Modernizr puder correr irá substituir a classe `.nojs` por `.js`, indicando que o *browser* utilizado suporta JavaScript. Desta

forma conseguimos manter o controlo sobre as capacidades do *browser* e utiliza-las a nosso favor. Este é o resultado dos testes realizados acedendo através de um *laptop* e utilizando o *browser* Chrome (na versão 26):

```
<html lang="en" class="js no-touch csstransforms csstransforms3d
csstransitions">
```

É desta forma que conseguimos tirar partido as classes `.csstransforms3d` e `.csstransitions` devolvidas pelos testes do Modernizr para criar a animação do movimento do menu à esquerda pretendida.

No caso do *browser* utilizado não suportar transformações e transições de CSS3, poderia utilizar-se a primeira abordagem, com o método `.animate` de jQuery. Para isso, seria novamente utilizado o Modernizr para chamar este método condicionalmente (na ausência das funcionalidades de CSS3 necessárias). No entanto, dado o mau desempenho verificado, a implementação reverte para a solução sem animação.

O resultado final está ilustrado na Figura 53.

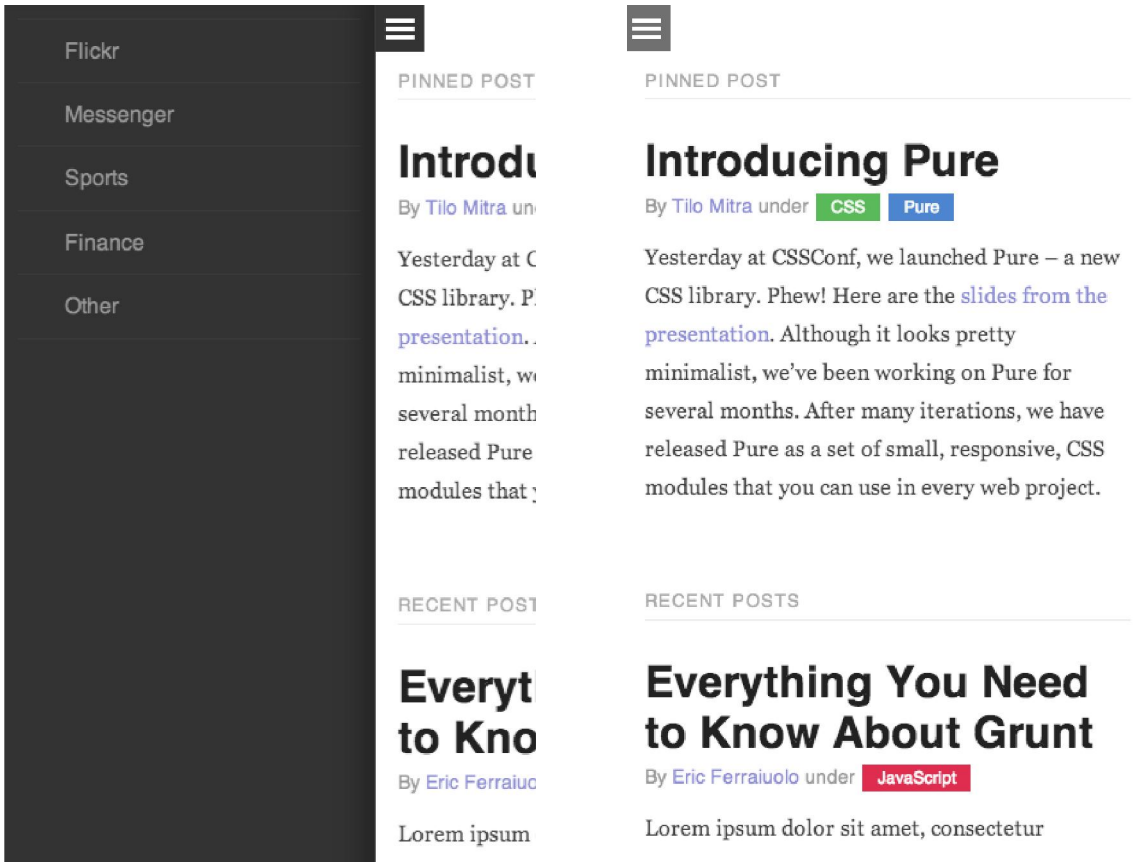


Figura 53. *Off-canvas* menu

Perante cada caso de adaptação concreto, cabe ao *designer* ou *developer* escolher qual das soluções padrão utilizar. Essa decisão poderá ser influenciada por alguns aspetos como o número de itens presentes no menu, a facilidade de integração com uma determinada abordagem em detrimento de outra, ou a forma como se adaptam os restantes elementos e componentes do *site*. Depois de escolhida uma das soluções padrão, basta incluir no projeto o pacote da respetiva solução, que contém um ficheiro JavaScript, CSS e Modernizr. O único requisito imposto para a utilização da ferramenta apresentada é que o menu tem se encontrar previamente definido, e com a estrutura composta por elementos HTML `ul` e `li`.

4.3 Otimizações

A forma como é apresentado o menu será sempre dependente do JavaScript estar ou não disponível no *browser* – estamos a lidar com *progressive enhancement*. Se JavaScript não for suportado, ou se por algum motivo não puder ser carregado, o utilizador recebe um *site* 100% funcional e com o conteúdo intacto. Se o JavaScript estiver disponível, conseguimos levar a experiência de utilização para um nível superior, com o menu adaptado ao tamanho do ecrã. Quanto mais capacidades foram suportadas pelo *browser*, melhor é a experiência de utilização oferecida ao utilizador.

Dos detalhes fornecidos pelo Modernizr sobre a capacidade do *browser* e dispositivo utilizado destaca-se também a informação sobre a interação através do toque. Os ecrãs tácteis convertem os eventos de clique em eventos de toque para assegurar que o comportamento do *site* se mantém consistente. No entanto, essa conversão resultava em cerca de 300 a 500ms de atraso. Utilizando o resultado do teste do `Modernizr.touch`, foi substituído o evento de clique (que despoletava a abertura do menu dos padrões *toggle* e *off canvas)*, por um evento de toque, quando é identificado num dispositivo ativo pelo toque. Deste modo, conseguimos otimizar a navegação conforme o dispositivo utilizado. Esta otimização tem como objetivo oferecer uma experiência de utilização mais adaptada ao tipo de dispositivo.

Para cada um dos padrões de menu foram feitas diferentes otimizações ao toque. No caso do menu *toggle*, o evento de clique foi substituído pelo evento *tap*, que permite um clique rápido nos dispositivos tácteis. Esta otimização permitiu remover o atraso motivado pela conversão do evento de clique, em aproximadamente meio segundo.

Para o padrão *off-canvas* recorreu-se também ao evento de *swipe*, que consiste num movimento curto e direcional, neste caso na horizontal. Este mecanismo pode também ser utilizado para aceder à navegação, resultando numa otimização que permite um rápido acesso ao menu, e que oferece uma experiência de utilização muito próxima das aplicações nativas de um sistema operativo móvel.

4.4 Desafios

Nas secções anteriores foi apresentada a ferramenta desenvolvida e explicados os detalhes da sua implementação. Durante a sua fase de desenvolvimento e de testes foram detetadas algumas limitações. Assim, esta secção trata os problemas identificados, que ao mesmo tempo consistem nos principais desafios encontrados ao longo das fases anteriormente mencionadas.

4.4.1 Identificação do elemento menu

Uma limitação identificada na implementação desta solução é a forma como é reconhecida a navegação, no caso de ser utilizada em *sites* pré-existentes. O reconhecimento da navegação limita-se a um conjunto de elementos e identificadores que podem não coincidir com o menu existente.

Uma maneira de contornar este problema será definir o atributo `data-role="navigation"` no elemento que contém a navegação. Desta forma, está garantida a identificação inequívoca do menu. No entanto, o objetivo desta implementação passa também por adaptar *sites* pré-existente, pelo que idealmente nenhuma alteração de HTML deveria ser feita.

4.4.2 Abordagem desktop down vs. mobile up

A criação de uma versão de menus adaptada a dispositivos móveis a partir de um menu já existente conduziu à utilização de uma estratégia baseada em *desktop down*, isto é, a adaptação é feita a partir do *site* tal como é apresentado em *desktop*, para ecrãs

sucessivamente mais pequenos. Esta é a estratégia que surgiu associada ao conceito de *responsive web design*.

No entanto, o trabalho de vários autores como (Wroblewski 2011) impulsionaram uma filosofia de adaptação reversa, *mobile up,* defendendo que esta é a abordagem mais acertada na construção de um *site*. Se até agora eram planeados *sites* para serem acedidos através de um browser de um computador, atualmente são projetados para serem acessíveis a partir de qualquer dispositivo, independentemente do seu tamanho. A ideia subjacente ao *mobile up* é partir do nível mais básico e com o menor espaço disponível, permitindo assim fazer uma abstração do que é essencial, e só depois ajustar o *design* para ecrãs sucessivamente maiores. Ainda assim, a adaptação dos menus teria que ser feita, só que pela ordem inversa. Ou seja, em vez de partirmos de uma navegação em linha, partiríamos de uma navegação adequada aos dispositivos com ecrã pequeno, que deveria transformar-se sucessivamente num menu para dispositivos com ecrãs maiores. Seguindo esta abordagem deixaríamos de poder suportar a adaptação de menu em *sites* existentes.

4.4.3 Integração com sites já existentes

A integração com *sites* já existentes envolve três fases: (i) escolha da abordagem a aplicar, (ii) adicionar os ficheiros necessários ao projeto e incluir a chamada para cada um deles nas páginas HTML relevantes do *site*, e (iii) fazer os ajustes de formatação necessários.

De *site* para *site* as técnicas e estratégias de desenvolvimento utilizadas podem ser muito variadas e até imprevisíveis. Este fator faz com que a integração com *sites* e aplicações já existentes nem sempre resulte, uma vez que é impossível prever todas as combinações possíveis. Para que a ferramenta seja capaz de suportar o maior número de *sites* possível, são permitidos alguns ajudes de formatação, que variam conforme o *site* e a abordagem escolhida.

Os ajustes podem ser feitos diretamente no ficheiro CSS do padrão escolhido. No entanto, a edição pode também ser feita na versão Less da folha de estilos. Aí, encontram-se declaradas variáveis de configuração, como por exemplo a cor de fundo da navegação, o tamanho de letra e as dimensões do ícone de menu, que podem ser facilmente modificadas pelo *designer/developer,* de acordo com cada *site*.

Por exemplo, foi definida uma variável para a cor de fundo da navegação, com o valor transparente para que se mantenha a cor da navegação original:

```
@backgroundColor:transparent;
@fontSize:inherit;
```

No entanto, pode facilmente definir-se uma cor específica para o fundo, e um tamanho de letra mais apropriado para as dimensões do ecrã, por exemplo:

```
@backgroundColor:#333;
@fontSize:12px;
```

A cor e o tamanho de letra anteriormente definidos nos valores das variáveis são aplicados a partir da especificação das seguintes regras de Less:

```
[data-role="navigation"] {
  background-color: @backgroundColor;
  font-size: @fontSize;
}
```

É também possível estabelecer o valor do comprimento da janela de visualização a partir do qual a adaptação se inicia (*breakpoint*):

```
@breakpointTablet: 768px;
@breakpointMobile: 480px;
@breakpoint: @breakpointTablet;

@media screen and (max-width: @breakpoint) { … }
```

Depois de editado o ficheiro com extensão Less, basta compilar esse código com auxílio de ferramentas como Visual Studio ou Less.js[30]. Como resultado, é gerado um novo ficheiro CSS atualizado e personalizado para um esse projeto.

A capacidade de adaptação varia conforme o *site* em questão, e é condicionada pela formatação utilizada. No entanto, o objetivo de auxiliar os *designers* e *developers* na obtenção de um menu capaz de se adaptar ao contexto do utilizador é cumprido.

[30] http://lesscss.org/ (acedido em 16/09/2013)

4.5 Conclusão

Neste capítulo introduziu-se uma ferramenta que permite transformar a navegação de *sites* seguindo os padrões de adaptação abordados na Secção 3.1.3. Baseia-se numa solução em JavaScript que gera uma versão adaptada da navegação, mediante o tamanho do ecrã, e tendo em conta as características do *browser* reportadas pelo Modernizr. Foram explicadas as decisões tomadas e apontados os problemas detetados, que surgem principalmente quando tentamos integrar a ferramenta com *site* pré-existentes. No entanto, mediante ajustes a integração é possível. No próximo capítulo são apresentados dois exemplos de teste da sua utilização.

Capítulo 5

Exemplos de utilização da ferramenta desenvolvida

Para garantir uma boa experiência de utilização é fundamental um sistema de navegação bem estruturado e o mais simples possível. No capítulo anterior foi apresentada a ferramenta desenvolvida ao longo deste trabalho, e que tem como objetivo ser uma componente auxiliar para se obter navegação adaptada ao tamanho do dispositivo do utilizador.

Na expetativa de perceber de que forma é possível integrar a ferramenta em diferentes *sites*, utilizamos dois *sites* para teste – um com navegação simples (com um único nível) e outro com navegação multi-nível. Ao longo deste capítulo são explicados os passos de integração da ferramenta para alcançar cada padrão de adaptação estudado, os problemas encontrados e ainda como podem ser ultrapassados para que os elementos que compõem a interface resultem funcional e visualmente bem.

5.1 Integração

Cada uma das soluções padrão exploradas para a adaptação de menus é constituída por dois ficheiros principais: um ficheiro JavaScript e um ficheiro CSS. Tal como vimos na Secção 4.2, o ficheiro JavaScript é responsável pela injeção de alguns elementos auxiliares e pela transformação de outros que pertencem ao menu. Por sua vez, o ficheiro CSS é responsável por formatar os menus adequadamente. Em conjunto, estes ficheiros permitem "desconstruir" a estrutura do menu existente e construir uma versão adaptada ao contexto do utilizador.

Para além dos ficheiros JavaScript e CSS incluídos em todas as abordagens, é também necessário incluir o *script* do Modernizr, para deteção do suporte de JavaScript e otimizar os eventos de toque, e ainda a biblioteca jQuery. O diagrama da Figura 54 ilustra um exemplo da organização dos ficheiros necessários (utilizando o padrão adaptativo *off-canvas*). Os ficheiros assinalados com (*) dependem do padrão de adaptação escolhido.

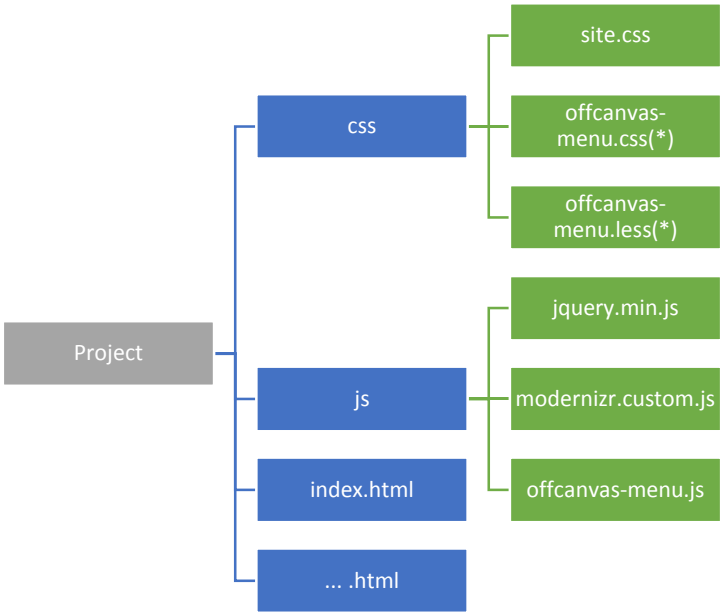


Figura 54. Organização dos ficheiros do projeto

O primeiro passo para utilizar a ferramenta é adicionar os ficheiros de CSS, Less e JavaScript nas pastas css e js do projeto do *site*. Depois de adicionar os ficheiros, basta importá-los em todos as páginas HTML, pela seguinte ordem (ver Figura 55):

1. Ficheiro CSS específico do *site* (site.css)
2. Ficheiro CSS de adaptação do menu (offcanvas-menu.css)
3. Chamada ao jQuery (jquery.min.js)
4. Chamada ao Modernizr (modernizr.custom.js)
5. Ficheiro JavaScript de adaptação do menu (offcanvas-menu.js)

Este procedimento é semelhante em todos os padrões de adaptação.

```
<head>
    <meta name="viewport" content="width=device-width"/>
    <link rel="stylesheet" href="css/site.css" type="text/css" />
    <link rel="stylesheet" href="css/offcanvas-menu.css" type="text/css" />
    <script type="text/javascript" src="js/jquery.min.js"></script>
    <script type="text/javascript" src="js/modernizr.custom.js"></script>
    <script type="text/javascript" src="js/offcanvas-menu.js"></script>
</head>
```

Figura 55. Chamadas aos ficheiros necessários

5.2 Navegação simples

A Figura 56 ilustra a navegação inicial de um dos *sites* usado para teste quando vista a partir de um computador e telemóvel. Este *site*, ainda em desenvolvimento, está a ser criado no âmbito do evento South by Southwest (SXSW), um festival de cinema, música e tecnologia que ocorre anualmente no Texas, nos Estados Unidos da América.

À medida que é reduzido o tamanho da janela de visualização, os itens do menu da Figura 56 vão-se reorganizado para ocupar o espaço disponível. No entanto, a reorganização natural do menu não é um procedimento escalável, o que faz com que o menu ocupe demasiado espaço se novos itens forem adicionados. Isso viola o princípio *content-first, nav-second* abordado na Secção 3.1.3.

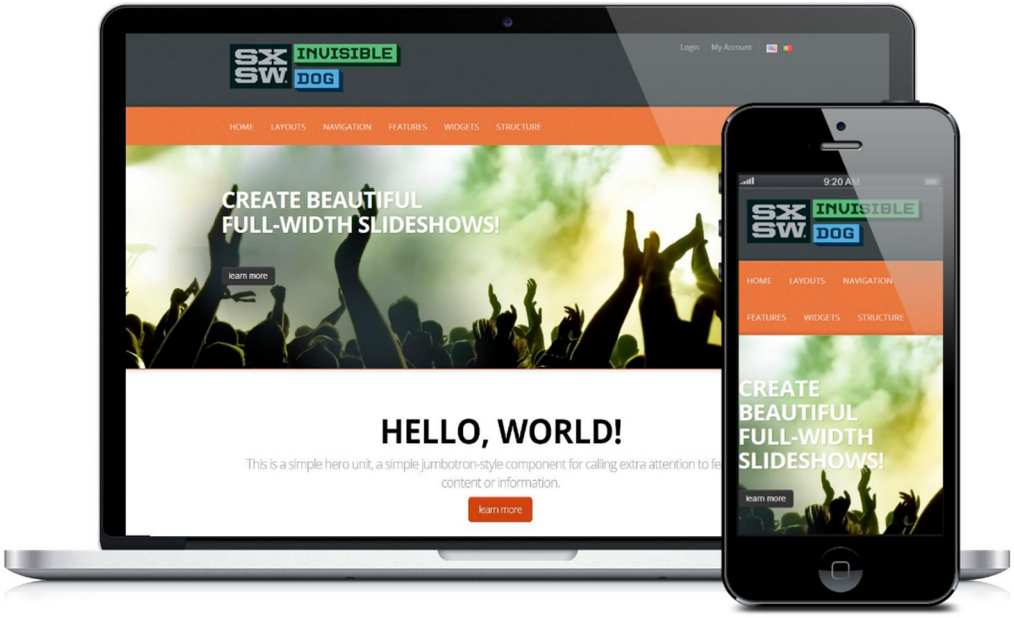


Figura 56. *Site* para teste com navegação simples

Como se pode ver na figura, a estrutura de navegação é formada por um único nível, com seis opções que conduzem para outras tantas páginas do *site*. A Figura 57 representa a organização da navegação.

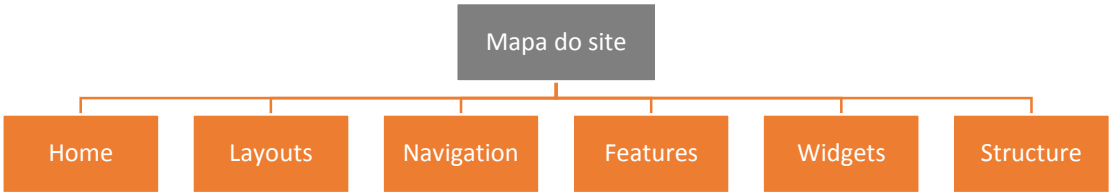


Figura 57. Mapa do *site*

O diagrama da Figura 58 ilustra a estrutura de pastas e ficheiros utilizados no caso deste *site* em particular. Aqui estão representados os ficheiros HTML que correspondem às várias páginas *web* e à *style sheet* do próprio *site* (site.css). Os ficheiros CSS e JavaScript responsáveis para a adaptação do menu (neste caso foi utilizada a técnica *select* menu) encontram-se distribuídos pelas pastas css e js, respetivamente.

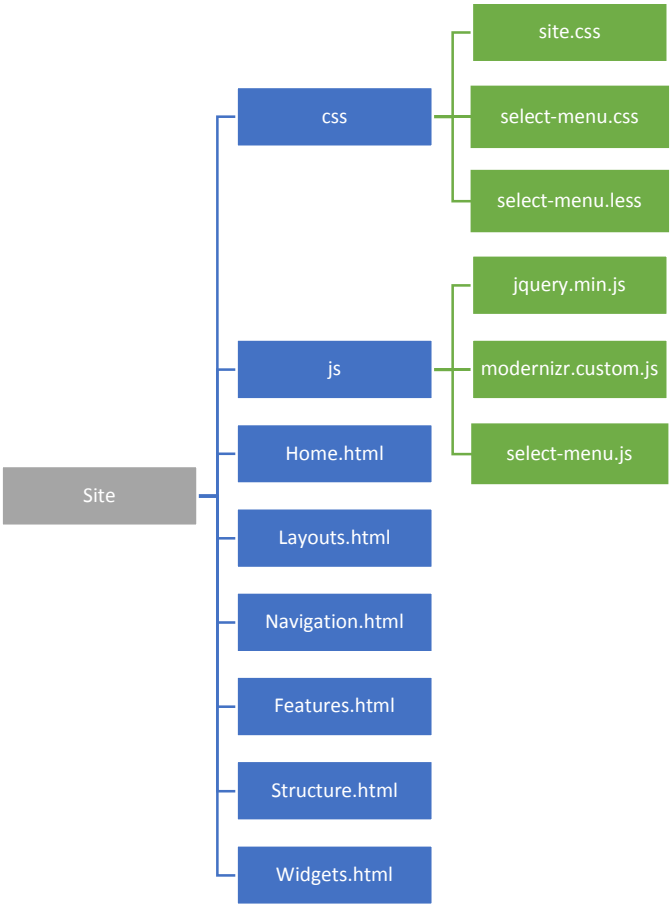


Figura 58. Estrutura de pastas e ficheiros a utilizar no padrão *select* menu

De seguida é exposto o resultado obtido com a integração da ferramenta com todos os padrões de adaptação implementados.

5.2.1 Select menu

O primeiro padrão testado foi o *select* menu. A estrutura das pastas e ficheiros utilizados está representada na Figura 58. A Figura 59 mostra o resultado da integração da ferramenta utilizando este padrão. A partir das variáveis de configuração foi possível ajustar ligeiramente o valor da margem do elemento *select*, para que este ficasse equilibrado (em relação à altura) na barra de menu. Assim, o código adicionado ao ficheiro select-menu.less foi o seguinte:

```
@margin: 5px 0 0 0;
```

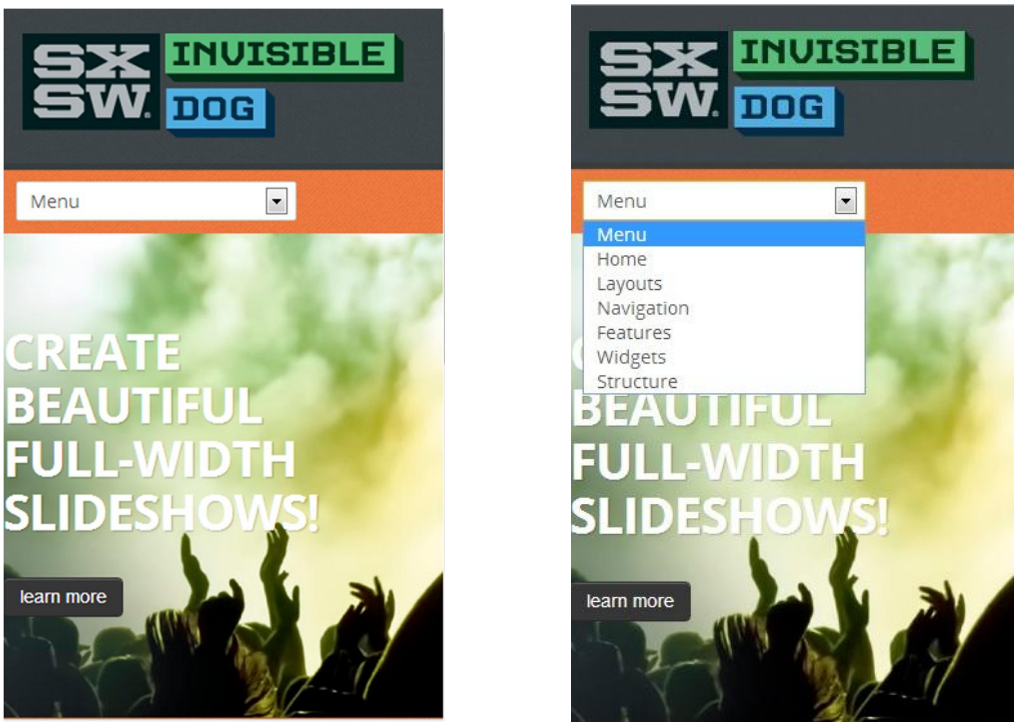


Figura 59. Resultado obtido utilizando a técnica *select* menu

5.2.2 Footer anchor menu

Este padrão foi o segundo a ser testado. Desta vez, os ficheiros incluídos foram footeranchor-menu.js e footeranchor-menu.css. Concluiu-se que a adaptação do menu depende

particularmente da formatação do rodapé do *site*. Os ajustes feitos evitam a herança de formatações presentes no rodapé. A alteração da cor de fundo e da cor dos itens de menu foram alguns desses ajustes. As variáveis adicionadas, e que sobrepõem as existentes por omissão, foram as seguintes:

```
@achorColor: white;
@anchorTextAlign: right;
@backgroundColor: #333;
@textColor: gray;
```

A Figura 60 ilustra o aspeto final obtido para a navegação no rodapé:

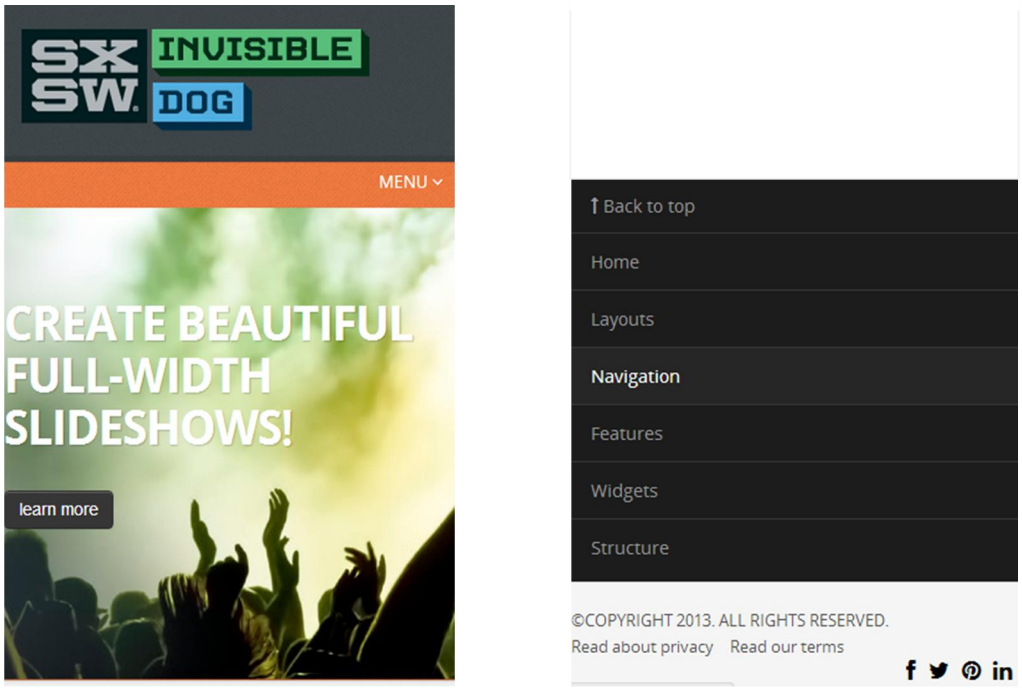


Figura 60. Resultado obtido utilizando a técnica *footer anchor* menu

5.2.3 Toggle menu

Para aplicar este padrão foram utilizados os ficheiros toggle-menu.js e toggle-menu.css que transformam e formatam a navegação. A cor de fundo foi herdada da navegação original do *site*. Foi utilizado um carácter especial para o ícone de fecho do menu e definido o seu tamanho, adicionando as seguintes variáveis ao ficheiro toggle-menu.less:

```
@iconClose: "&#10005;";
@iconSize: 18px;
```

A Figura 61 ilustra o resultado obtido.

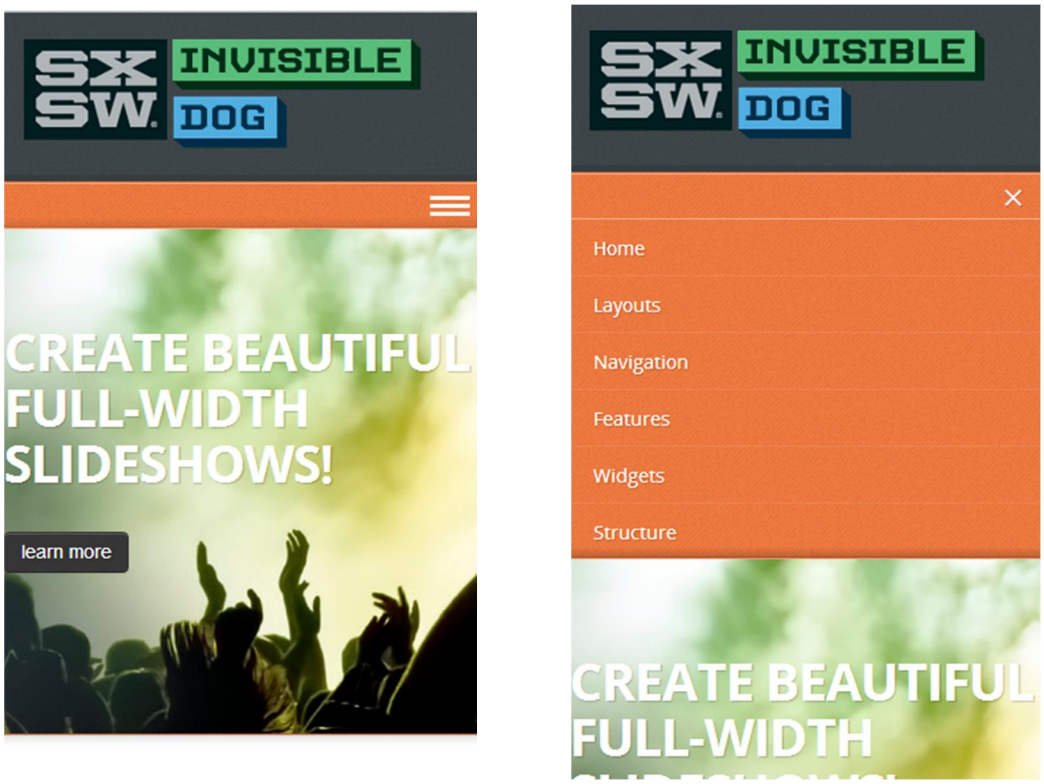


Figura 61. Resultado obtido utilizando a técnica *toggle* menu

5.2.4 Off-canvas menu

Para a integração do *off-canvas* menu foi necessário incluir uma vez mais os ficheiros JavaScript e CSS (offcanvas-menu.js e offcanvas-menu.css) responsáveis pela injeção de novos elementos e pela formatação dos mesmos. A utilização deste padrão no *site* resultou na necessidade de ajustar algumas das propriedades do ícone de menu (identificadas com o prefixo @icon) e também as cores de fundo e do texto dos itens da navegação. Para obter essas alterações foram adicionadas as seguintes variáveis, cujos valores sobrepõem os valores definidos por omissão:

```
@iconWidth:25px;
@iconHeight:25px;
@iconBackground: #333;
@iconOpacity: 0.5;
@background: #333;
@color: gray;
```

A Figura 62 mostra o resultado obtido.

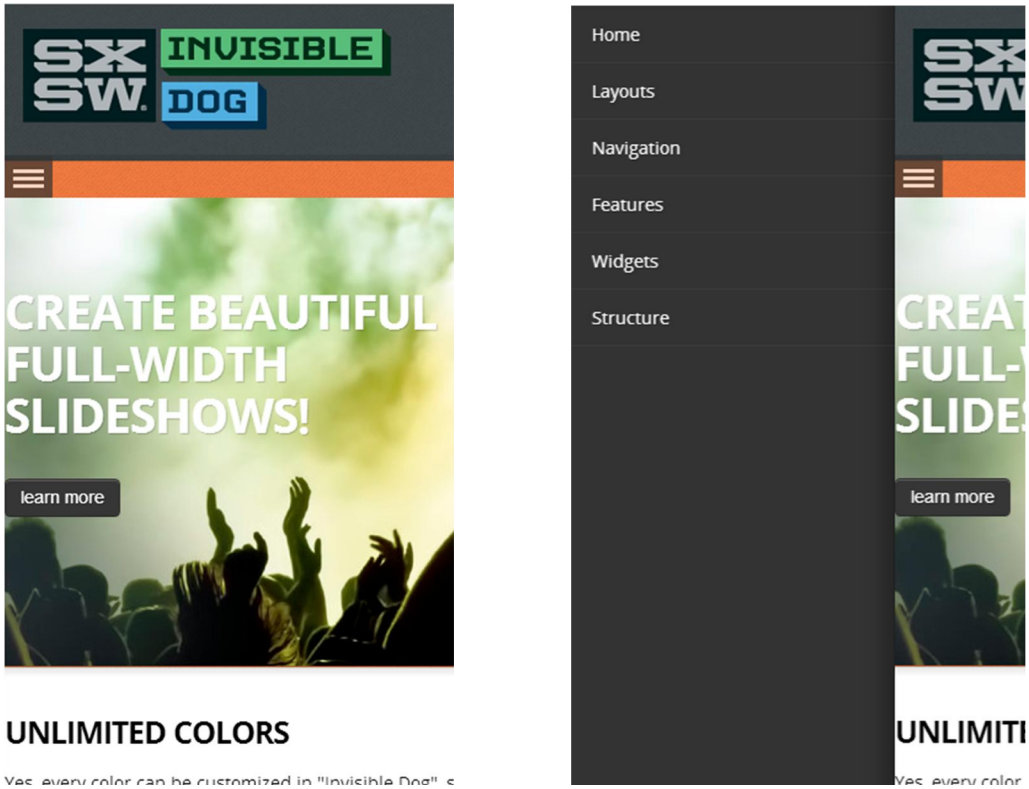


Figura 62. Resultado obtido utilizando a técnica *off-canvas* menu

5.3 Navegação multi-nível

Até agora vimos como é que a ferramenta desenvolvida pode ser utilizada em sistemas de navegações simples, com apenas um nível de estrutura. No entanto, a realidade atual é também constituída por *sites* com estruturas de navegação exibindo vários níveis de profundidade. De forma a ilustrar as quatro variantes de adaptação de menus num *site* com navegação de vários níveis, foi utilizado o site apresentado na Figura 63. Este *site*, ainda em fase de desenvolvimento, está a ser construído para uma clínica de serviços de saúde.

Figura 63. *Site* para teste com navegação multi-nível

Em dispositivos como *desktops* e *laptops*, o espaço é suficiente para dispor os itens de menu numa única linha horizontal. No entanto, em dispositivos com ecrã de tamanho pequeno, como é o caso dos *smartphones*, o espaço disponível é consideravelmente menor, e portanto o menu ocupa demasiado espaço no ecrã e a subnavegação deixa de ser acessível.

O diagrama da Figura 64 ilustra a estrutura da navegação do *site*. Esta é composta por três níveis de profundidade. O primeiro nível da navegação é constituído por sete itens. Três desses sete itens (*Health Services*, *Centers of Care* e *Pacients and Visitors*) possuem submenus. O item *Pacients and Visitors,* desdobra-se em dois subitens (*Pacients* e *Visitors*), cada um deles com subitens (terceiro nível de navegação).

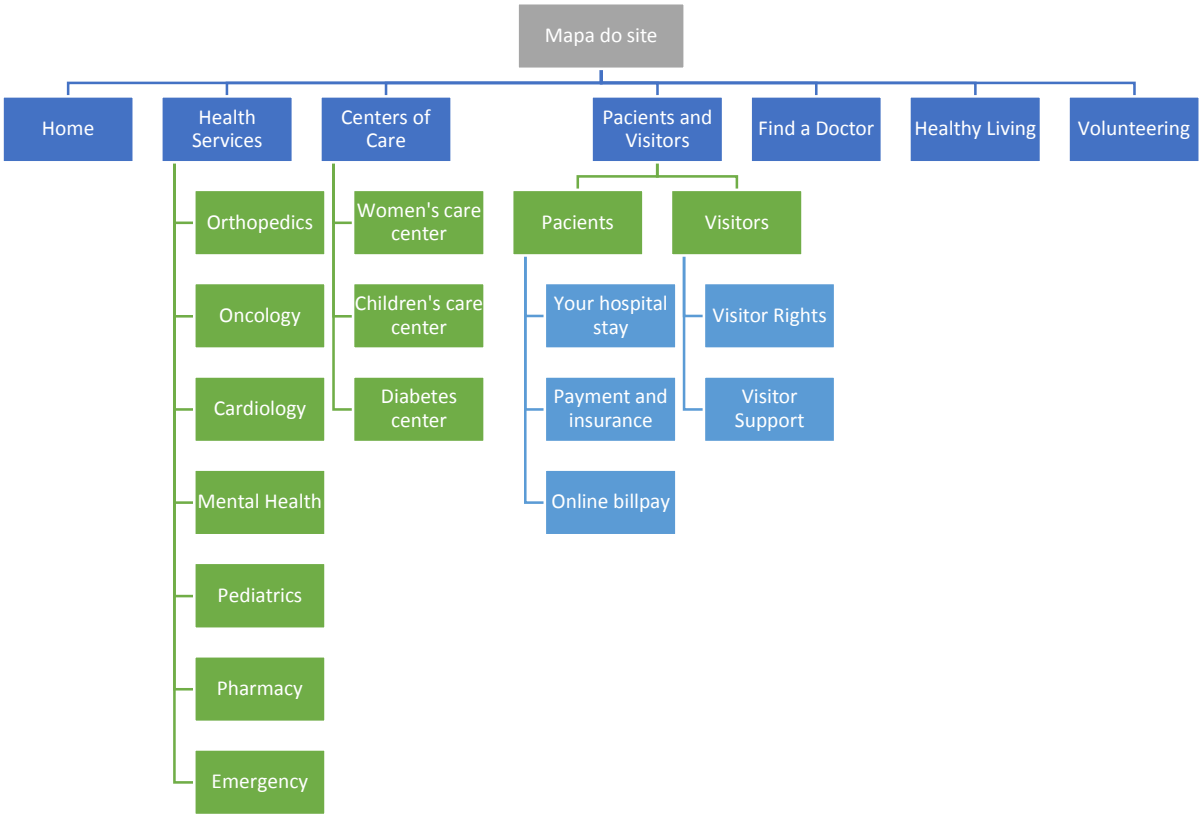


Figura 64. Mapa do *site*

O resultado da integração com a ferramenta, utilizando cada uma das técnicas de adaptação é apresentado de seguida. Para que a formatação do menu ficasse enquadrada com o *site* o mais possível, foram mais uma vez utilizadas as variáveis de configuração para fazer vários ajustes de formatação. A configuração utilizada é também apresentada a seguir.

5.3.1 Select menu

O primeiro padrão a ser testado foi o *select* menu. Para que o menu ficasse mais enquadrado com o *site*, foram configuradas as cores e a altura do elemento *select*. Assim, foram adicionadas ao ficheiro select-menu.less as seguintes variáveis:

```
@backgroundColor: #1baefb;
@textColor: white;
@height: 50px;
```

As restantes variáveis foram mantidas com o seu valor por omissão. O resultado final da integração deste padrão encontra-se na Figura 65. Note-se que para se distinguirem os subníveis, é utilizado o carácter "-".

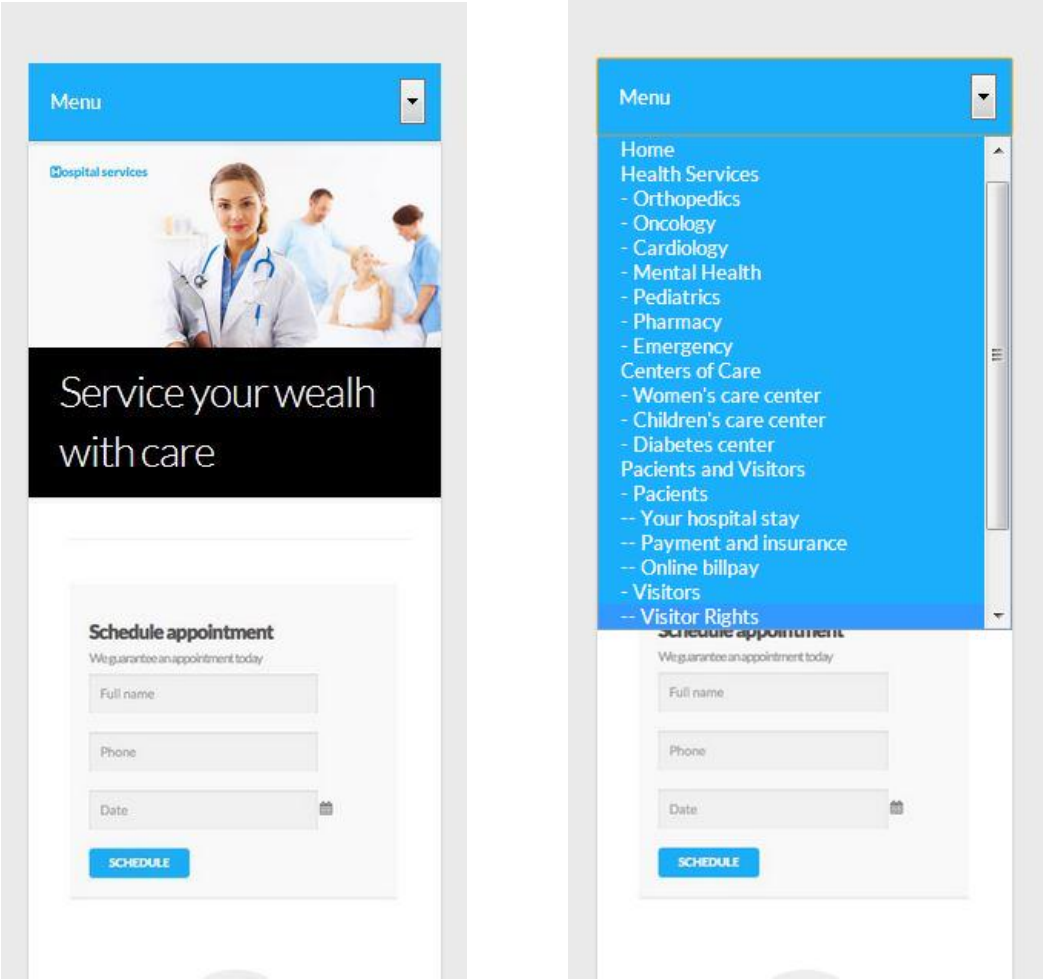


Figura 65. Resultado obtido utilizando a técnica *select* menu

Explorando um pouco mais as variáveis de configuração, podemos obter um aspeto diferente para o *select* menu, e assim fugir ao aspeto do elemento *select* tradicional. Por exemplo, se adicionarmos ao ficheiro select-menu.less um ícone de menu e definirmos que o elemento *select* é transparente (opacidade com valor 0), conseguimos obter o resultado apresentado na Figura 66.

```
@icon: "\f0c9";
@iconSize: 25px;
@opacity: 0;
```

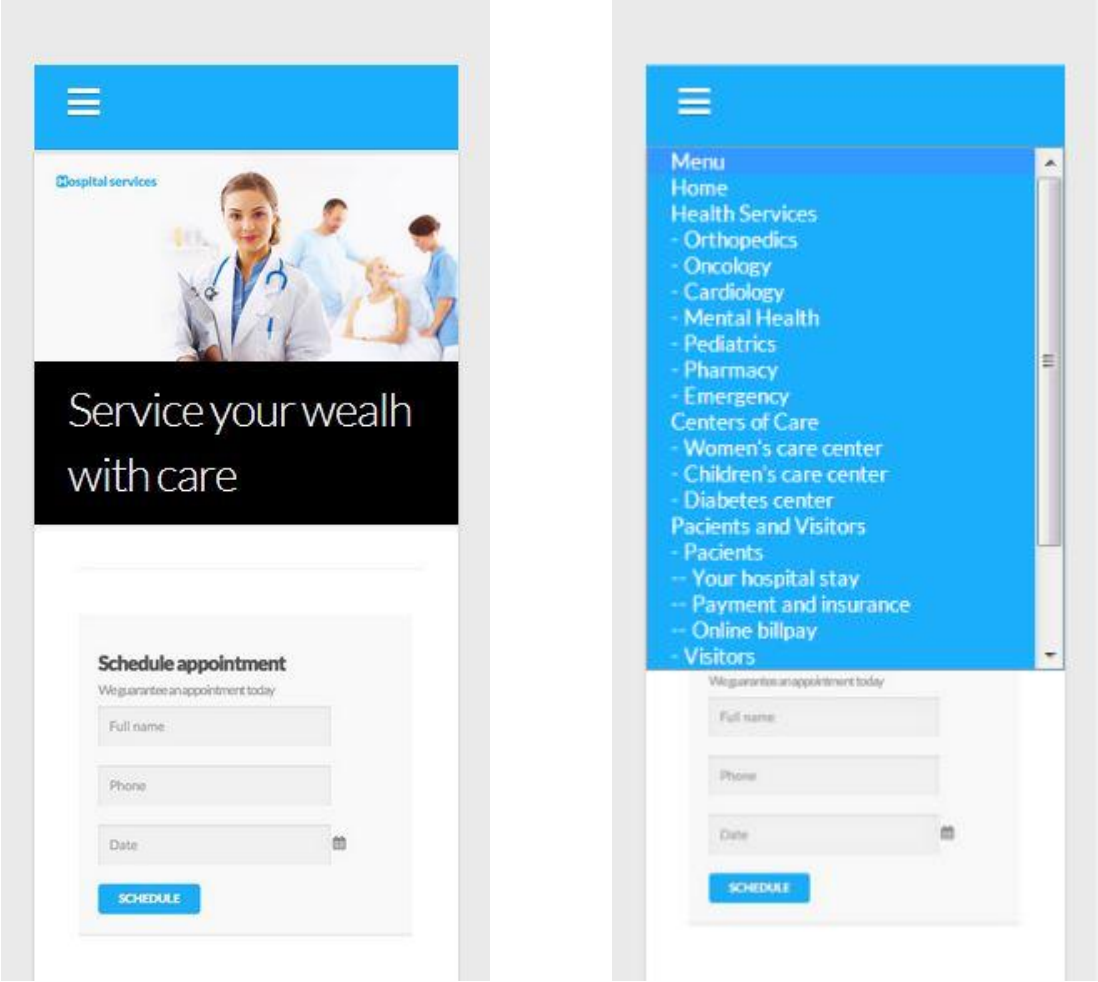


Figura 66. Resultado obtido utilizando a técnica *select* menu (ajustes adicionais)

5.3.2 Footer Anchor

O *footer anchor* foi o padrão seguinte a ser testado. Nesta técnica de adaptação, os diferentes níveis de formatação são distinguidos com diferentes valores de indentação. Para este caso foi necessário configurar, através das variáveis Less, o elemento que serve de âncora para o menu. As variáveis que guardam a formatação desse elemento são facilmente identificadas pelo prefixo @anchor. Além disso, foi também definida a cor do fundo e do texto da navegação,

para sobrepor a formatação atribuída ao rodapé do *site*. Foram adicionadas as seguintes varáveis ao ficheiro footeranchor.less:

```
@anchorBackground: #1baefb;
@achorColor: white;
@anchorPadding: 0;
@anchorHeight: 50px;
@anchorLineHeight: 50px;
@anchorTextAlign: center;
@anchorFontSize: 15px;
@backgroundColor: #1baefb;
@textColor: white;
```

A Figura 66 ilustra o resultado obtido.

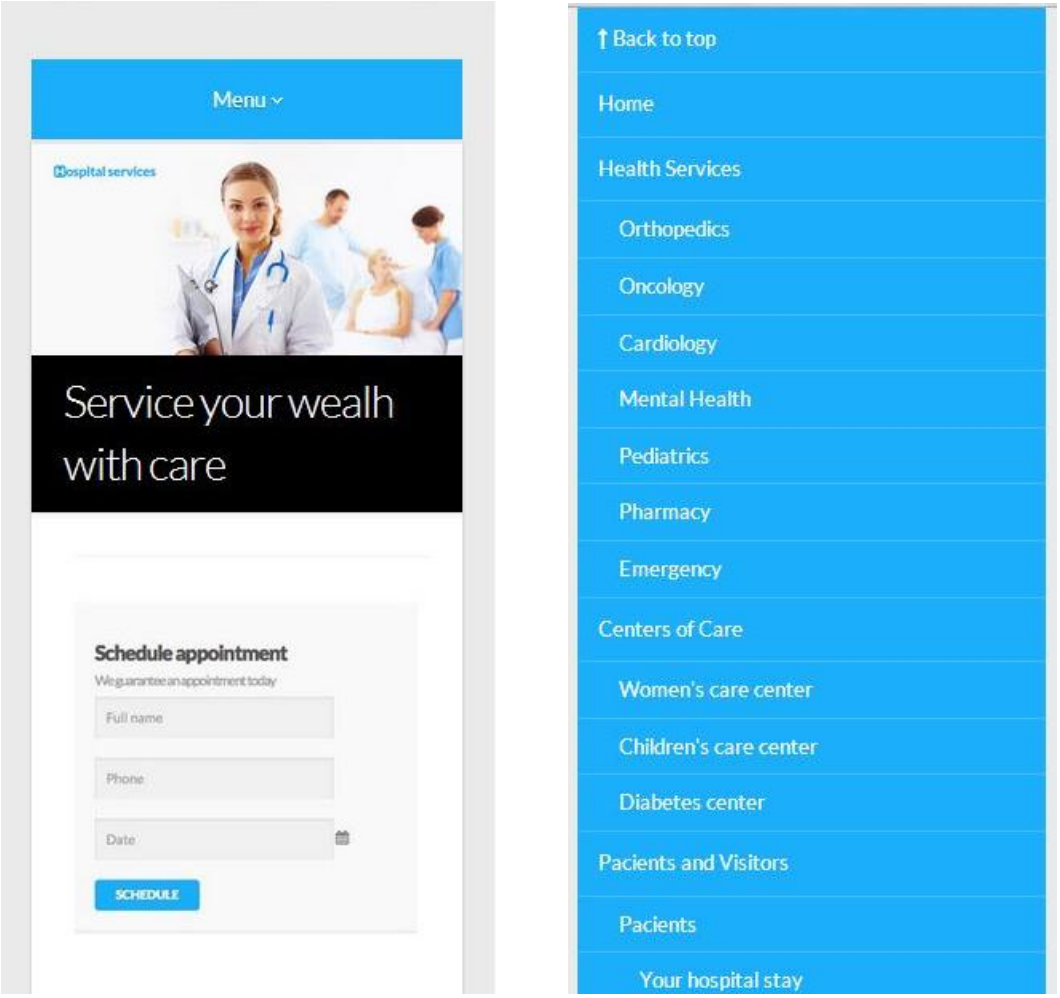


Figura 67. Resultado obtido utilizando a técnica *footer anchor* menu

Para navegações mais complexas esta abordagem de menu no rodapé deixa de ser apropriada. Na tentativa de evitar que a navegação não se torne demasiado extensa, poderia optar-se por uma técnica capaz de expandir e retrair os subitens do menu, à semelhança do que acontece no padrão *toggle.*

5.3.3 Toggle menu

Nesta abordagem, quando o menu é aberto são expandidos apenas os itens do primeiro nível da navegação. Se um item possui subitens, é desenhada uma seta que expande a respetiva subnavegação.

Para este caso optou-se por utilizar o mesmo ícone de abrir e fechar para o menu. Optou-se também por distinguir os diferentes níveis de profundidade com diferentes cores. A cor do texto é determinada automaticamente a partir do contraste em relação à cor de fundo. As variáveis de configuração foram adicionadas ao ficheiro toggle.less com os seguintes valores:

```
@iconOpen: "\f0c9";
@iconClose: "\f0c9";
@iconSize: 16px;
@iconBackground: #1baefb;
@backgroundFirstChild: #1baefb;
@backgroundSecondChild: #cfeffe;
@backgroundThirdChild: #fff;
```

A Figura 68 mostra o aspeto final da navegação ilustrando duas situações distintas: quando o menu se encontra fechado, e quando apresenta apenas o primeiro nível de navegação visível.

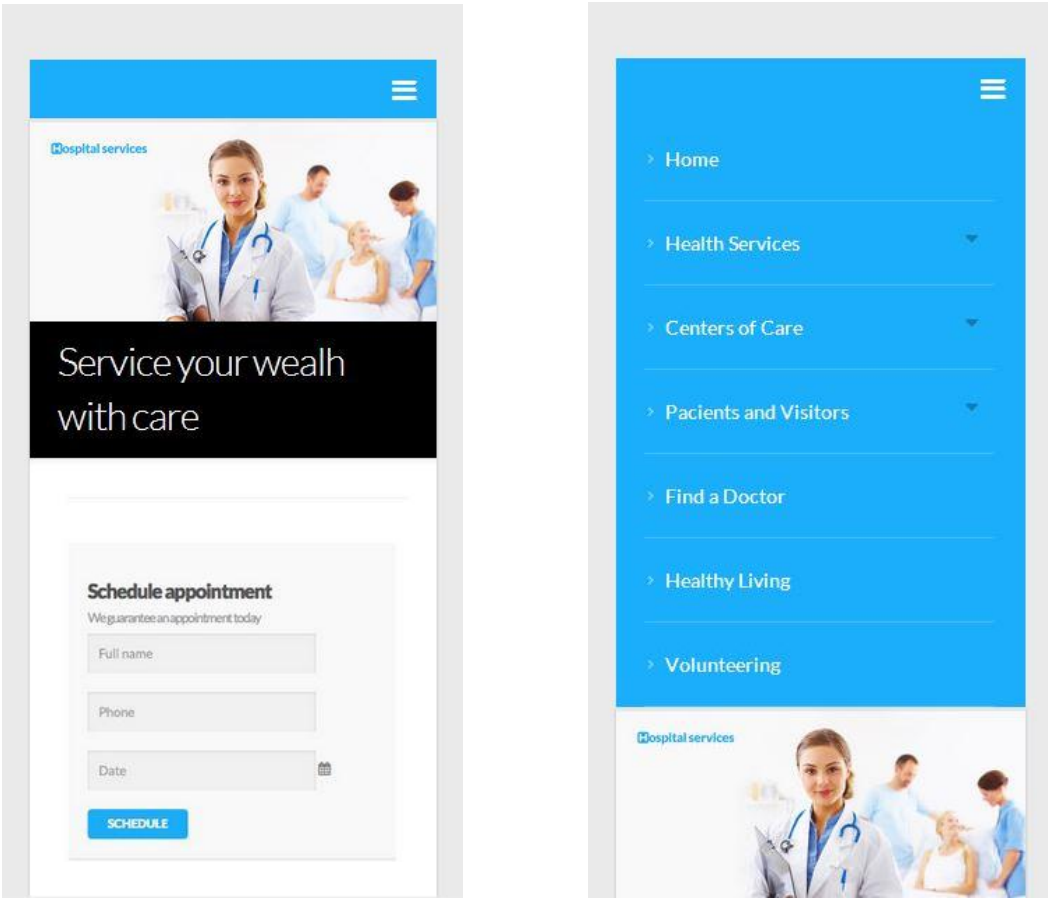


Figura 68. Resultado obtido utilizando a técnica *toggle* menu

A Figura 69 mostra um exemplo da navegação com o segundo e o terceiro nível visíveis.

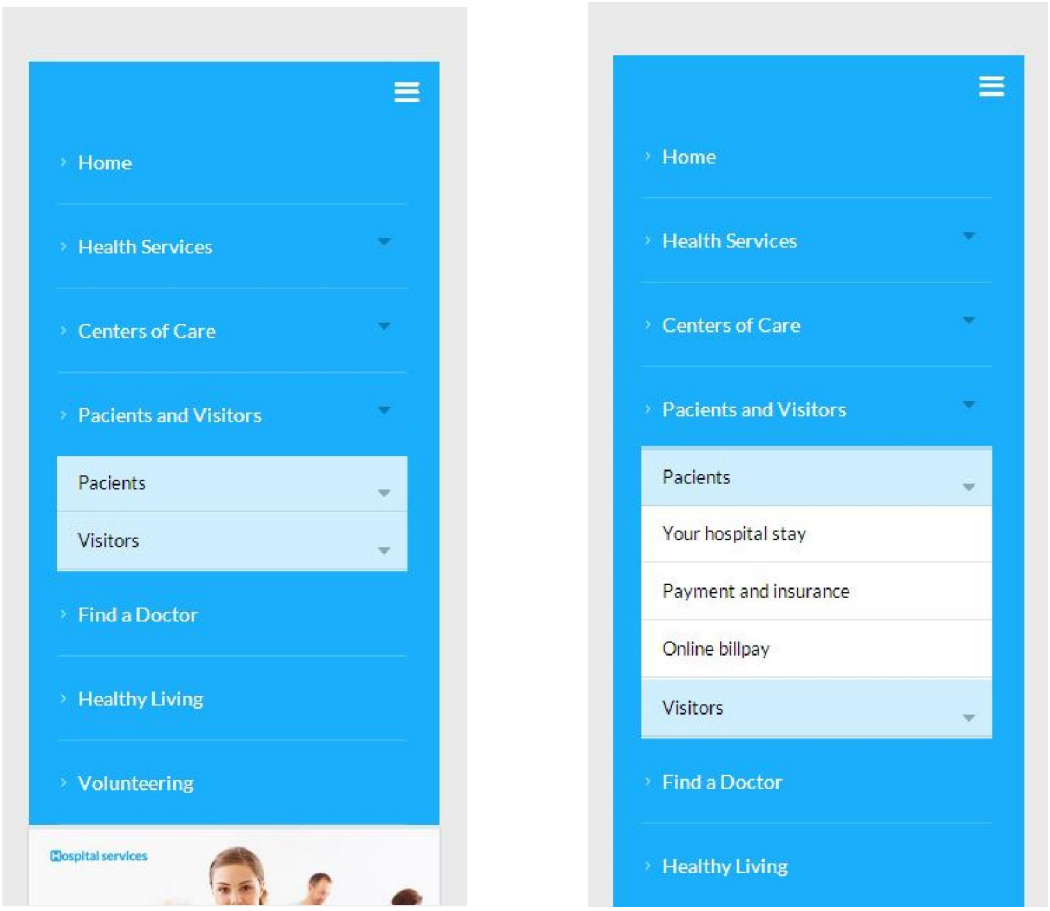


Figura 69. Resultado obtido utilizando a técnica *toggle* menu (2º e 3º nível expandidos)

5.3.4 Off-canvas menu

O último teste foi feito com o padrão *off-canvas*. O resultado da integração utilizando este padrão pode ser encontrado na Figura 70. Diferentes níveis de navegação são distinguidos pela sua indentação. No entanto, foram também usadas diferentes cores do fundo (definidas nas variáveis @background e @backgroundFirstChild), com o objetivo de destacar os itens do primeiro nível. Foi ainda configurado o tamanho do ícone de menu (prefixo @icon), a cor e a percentagem de opacidade. Assim, as variáveis adicionadas ao ficheiro Less foram as seguintes:

```
@iconWidth:27px;
@iconHeight:22px;
@iconBackground: #1baefb;
@iconOpacity: 0.5;
@backgroundFirstChild: #cfeffe;
@background: white;
```

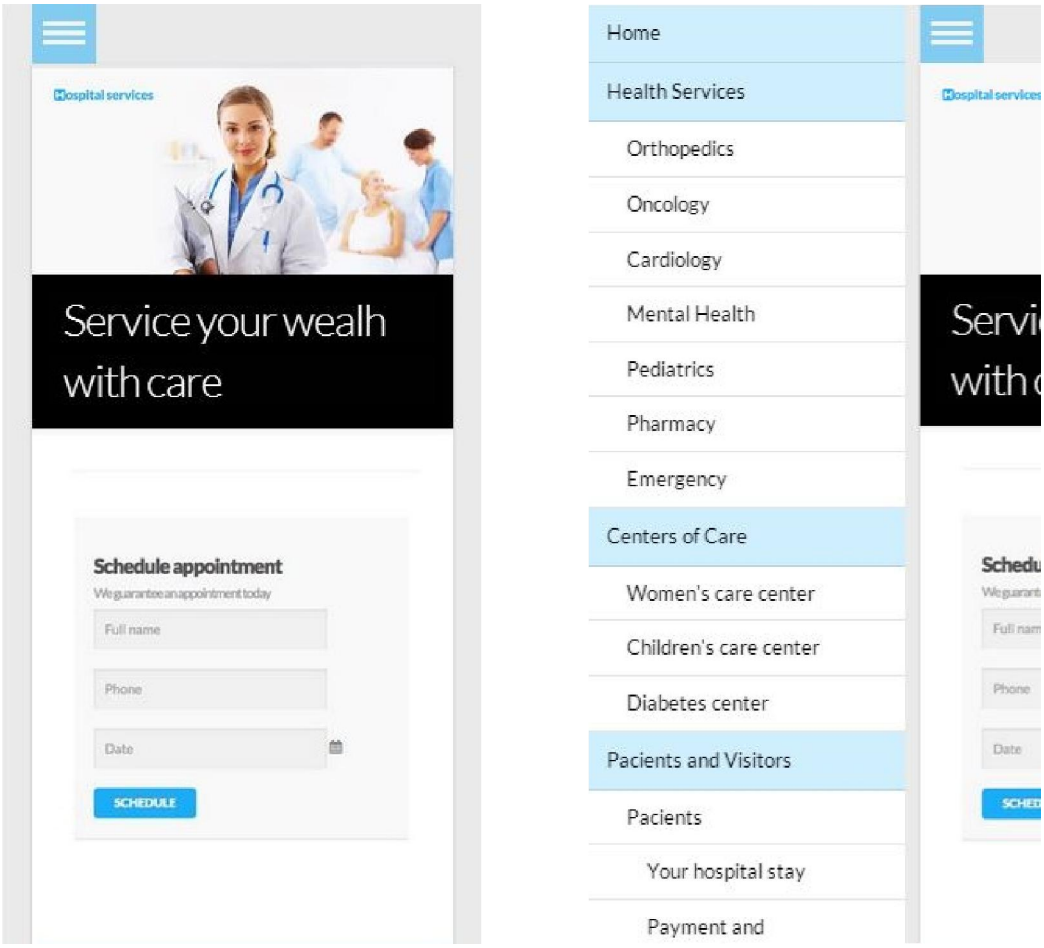


Figura 70. Resultado obtido utilizando a técnica *off-canvas* menu

5.4 Conclusão

Geralmente, em dispositivos com ecrã reduzido não existe espaço suficiente para dispor os itens de menu numa única linha horizontal. Por isso, nestes casos é necessário recorrer a um dos padrões de adaptação para navegação. Todos os padrões permitem aceder facilmente à navegação e reduzem consideravelmente o espaço ocupado.

Manter a adaptação natural dos elementos da navegação pode ser uma alternativa simples e adequada para menus com um número reduzido de itens (três ou quatro itens). Poderia optar-se por este tipo de adaptação na navegação da Figura 56, mas a garantia de que o menu não poderia ser escalável induz a que se deva recorrer a um dos padrões adaptativos. Como se trata de uma navegação simples, com um só nível, todos os padrões resultaram bem.

O padrão que adapta o menu num *select* menu é uma alternativa que funciona independentemente do número de elementos que a navegação contem. No entanto, quando temos múltiplos níveis de navegação, pode não ser intuitivo perceber a hierarquia existente (ver Figura 65).

Por sua vez, a abordagem *footer anchor* põe em prática o princípio *content-first, nav-second* – o conteúdo surge em primeiro lugar, e a navegação surge depois, no rodapé (ver Figura 60). No entanto, esta abordagem pode torna-se desorientadora para os utilizadores que não esperam ser conduzidos até ao rodapé do *site* quando tentam consultar a navegação. Além disso, este padrão fica demasiado longo quando tem muitos itens de menu (ver Figura 67).

Com a adaptação segundo o padrão *toggle*, podemos aceder à navegação e escondê-la por trás de um ícone de menu. Os itens da navegação descem no mesmo sítio onde o ícone se encontra (ver Figura 61), logo não provocam desorientação ao utilizador. Para um menu com vários níveis de profundidade, como é o caso da Figura 68, esta técnica também resulta bastante bem: os subitens permanecem escondidos quando a navegação é acedida (ver Figura 69), evitando que se torne uma lista demasiado longa como acontece com a técnica *footer anchor* da Figura 67.

Por fim, a adaptação segundo o padrão *off-canvas* é uma solução que também funciona para menus com um (Figura 62) ou mais níveis (Figura 70). Por ser utilizada frequentemente em aplicações móveis nativas, o seu funcionamento deverá ser já familiar para grande parte dos utilizadores.

Capítulo 6

Conclusões

A construção de *websites* segundo os princípios do *responsive web design* vem incentivar a difusão de novas experiências de utilização adequadas à variedade de dispositivos existentes. Este fator tem um impacto particularmente benéfico para as pequenas e médias empresas, que desta forma conseguem chegar aos utilizadores que usam outros canais de acesso à *web* que não o computador.

O *responsive web design* tem como foco principal melhorar a experiência de utilização nos diferentes dispositivos, desde o computador *desktop* e *laptop*, *tablet* e telemóvel. No entanto, é importante salientar que *responsive* não é significa construir versões limitadas de *sites* para dispositivos mais pequenos. Pelo contrário, os *sites* devem estar completos e a sua informação deve manter-se acessível, qualquer que seja o contexto do utilizador e as capacidades do *browser* utilizado. Para isso, é necessário encontrar estratégias de adaptação capazes de responder a diferentes contextos. Algumas dessas estratégias podem ser reconhecidas como soluções padrão, e foram exploradas ao longo do Capítulo 3.

No entanto, *responsive web design* vai para além de adaptações ao nível da interface. Para progredir na experiência oferecida ao utilizador, podem ser criadas soluções de otimização para dispositivos-alvo com um determinado conjunto de características. Por exemplo, otimizar para dispositivos cuja interação é feita através do toque, ou definir os recursos descarregados com base em determinadas variáveis conforme o ambiente em que o utilizador se encontra, trazendo melhorias em termos de desempenho. Para além de melhorar a experiência de utilização ao permitir navegar em páginas *web* mais rápidas, utilizar apenas os recursos necessários e os mais adequados para cada contexto traz outras vantagens associadas, como diminuir a largura de banda utilizada e preservar a bateria, aspetos particularmente importantes em dispositivos móveis.

6.1 Objetivos atingidos

Um dos objetivos atingidos consistiu no estudo de um modelo para interfaces *web* capazes de se adaptarem aos diferentes contextos de utilização, em que o contexto corresponde ao tamanho do ecrã do dispositivo. Assim, conclui-se que para diferentes contextos é possível usar uma única interface final, mesmo desenhando diferentes interfaces concretas. Na interface final é depositada toda a lógica de adaptação, evitando recorrer a várias interfaces finais, uma para cada contexto.

Um objetivo proposto foi o de desenvolver uma ferramenta que permitisse a adaptação da navegação em *sites* já existes, libertando o *designer* ou *developer* dessa tarefa que é inevitável em *sites responsive*. Seguindo a abordagem referida acima, a adaptação é feita ao nível da interface final. Foi implementado um conjunto de soluções padrão para a adaptação de menus, e a escolha da abordagem a utilizar é feita pelo utilizador, decidindo qual o mais adequado para a sua aplicação.

6.2 Trabalho futuro

Como trabalho futuro, espera-se conseguir validar juntos dos utilizadores quais os padrões de adaptação que são melhor aceites e os que, pelo contrário, causam mais frustração ao utilizador. Para isso, será necessário definir, em primeiro lugar, um conjunto de métricas capazes de avaliar a preferência dos utilizadores.

Propõem-se ainda melhorar a ferramenta desenvolvida. A sua limitação reside na capacidade de adaptação que varia conforme o *site* em questão, uma vez que é condicionada pela formatação já existente. Com o objetivo de ultrapassar esta limitação, é necessário continuar com o processo de testar a integração da ferramenta num conjunto alargado de *sites*, para que se consiga obter uma solução mais genérica. Tal solução permitirá também minimizar consideravelmente a necessidade de ajustes adicionais na utilização da ferramenta. Assim que que subsistir uma versão bastante estável da ferramenta, pretende-se disponibiliza-la livremente, e pedir o *feedback* de possíveis utilizadores (*beta testers*).

A ferramenta dirige-se tanto para *sites* novos como para *sites* pré-existentes. Por isso, é muito importante que o resultado obtido com a utilização da ferramenta fique enquadrado com o *design* presente no *site* em que é aplicada. A implementação de um customizador *online* que

permitisse configurar certas caraterísticas, como por exemplo o tamanho da letra, cores de fundo e o ícone de menu utilizado, seria interessante sob o ponto de vista do utilizador. Seria também uma forma de evitar a necessidade de editar manualmente as variáveis de configuração.

Referências

Ates, Faruk. 2010. "Taking Advantage of HTML5 and CSS3 with Modernizr." *A List Apart*, June 22: URL: http://alistapart.com/article/taking-advantage-of-html5-and-css3-with-modernizr.

Bulger, Danielle. 2010. "Smartphone Owners: A Ready and Willing Audience." *Compete Pulse.* March 12. URL: https://blog.compete.com/2010/03/12/smartphone-owners-a-ready-and-willing-audience/.

Cremin, Ronan, and Luca Passani. 2012. "Server-Side Device Detection: History, Benefits And How-To." *Smashing Maganize.*

Dale, Tom. 2013. "Progressive Enhacement." *Tom Dale.* September 2. URL: http://tomdale.net.

Filament Group. 2012. *Respond.js: Fast CSS3 Media Queries for Internet Explorer 6-8 and more.* Agosto 02. URL: http://filamentgroup.com/lab/respondjs_fast_css3_media_queries_for_internet_explorer_6_8_and_more.

Findlater, L., and J. Mcgrenere. 2008. "Impact of screen size on performance." *Proceeding of the Twenty-sixth Annual SIGCHI Conference on Human Factors in Computing Systems.* New York. 1247–1256.

Frost, Brad. 2012. "Beyond media queries: anatomy of an adaptive web design." *Smashing Conference.*

Frost, Brad. 2012. "Creating a Mobile-First Responsive Web Design." *HTML5 Rocks (Tutorials)* URL: http://www.html5rocks.com/en/mobile/responsivedesign/.

Frost, Brad, interview by John Macpherson. 2013. *Mobile Navigation and Patterns*

Frost, Brad. 2012. "Responsive Navigation Patterns." *Brad Frost Web.* February 24. URL: http://bradfrostweb.com/blog/web/responsive-nav-patterns/.

Gajos, Krzysztof Z., Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. 2006. "Exploring the Design Space for Adaptive Graphical User Interfaces." *Proceedings of Advanced Visual Interfaces.*

Graeve, Katrien De. 2011. "Responsive Web Design." *MSDN Magazine.* http://msdn.microsoft.com/en-us/magazine/hh653584.aspx.

Grigsby, Jason. 2013. "Responsive Web Design is Solid Gold." *Cloud Four Blog* URL: http://blog.cloudfour.com/responsive-web-design-is-solid-gold/.

Gustafson, Aaron. 2011. *Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement.* Tennessee, USA: Easy Readers.

Hannemann, Anselm. 2013. "Responsive design: we are not there yet." *.Net Magazine.*

Hixon, Jeremy. 2011. "An Introduction To LESS, And Comparison To Sass." *Smashing Magazine.*

Internet World Stats. 2012. "World Internet Users." *Internet World Stats.* June 30. URL: http://www.internetworldstats.com/stats.htm.

Jankord, Brett. 2012. *Categorizr – A modern device detection script.* January 16. URL: http://www.brettjankord.com/2012/01/16/categorizr-a-modern-device-detection-script/.

Kadlec, Tim. 2013. *Implementing Responsive Design: Building sites for an anywhere, everywhere web.* Berkeley, California: New Riders.

Kadlec, Tim, interview by John Macpherson. 2013. *Site Performance*

La, Nick. 2010. "CSS3 Media Queries." *Web Designer Wall.* August 18. URL: http://webdesignerwall.com/tutorials/css3-media-queries.

Lavie, Talia, and Joachim Meyer. 2010. "Benefits and costs o fadaptive user interfaces." *SienceDirect* 508–524.

Marcotte, Ethan. 2009. "Fluid Grids." *A List Apart Magazine.*

Marcotte, Ethan. 2010. "Fluid Images." *Unstoppable Robot Ninja* URL: http://unstoppablerobotninja.com/entry/fluid-images/.

Marcotte, Ethan. 2010. “Responsive Web Design.” *A List Apart Magazine.*

Marcotte, Ethan. 2011. *Responsive Web Design.* Edited by Mandy Brown. New York: Jeffrey Zeldman.

Model-based User Interfaces Incubator Group. 2010. "Incubator Group Report." *W3C.* January 17. Accessed Outubro 10, 2013. http://www.w3.org/2005/Incubator/model-based-ui/wiki/Incubator_Group_Report.

Mohomed, Iqbal, Adin Scannell, Nilton Bila, Jin Zhang, and Eyal de Lara. 2007. "Correlation-Based Content Adaptation." *International Federation for Information Processing* 101–120.

Moore, Jordan. 2013. "Responsible Considerations For Responsive Web Design." *Smashing Magazine.*

Nebeling, Michael, Maximilian Speicher, and Moira C. Norrie. 2013. "CrowdAdapt: Enabling Crowdsourced Web Page Adaptation for Individual Viewing Conditions and Preferences." *EICS 2013.* London.

Oliveira, Beatriz. 2013. "Responsive Web Design and beyond." *Orchard Harvest.*

Pew Research Center. 2012. "Future of mobile news." *Journalism.org.*

Pietrusky, Tim. 2012. "Responsive Menu Concepts." *Appliness* 109-123.

Rivoal, Florian. 2012. "Media Queries." *W3C Recommendation.*

Rothrock, L., R. Koubek, F. Fuchs, M. Haas, and G. Salvendy. 2002. "Review and reappraisal of adaptive interfaces: toward biologically." *Theoretical Issues in Ergonomics Science 6 (2)* 157–172.

Royal Pingdom. 2013. *Internet 2012 in numbers.* January 16. URL: http://royal.pingdom.com/2013/01/16/internet-2012-in-numbers/.

Sampaio, Ana. 2013. "Fluid Typography for Responsive Websites." *Bind Tuning Academy.* June 18. URL: http://academy.bindtuning.com/fluid-type-for-responsive-websites/.

Sena, Pete. 2013. "Beyond responsive design: discover context-first." *Creative Bloq.* August 30. URL: http://creativebloq.com.

Silva, J. L., O. R. Ribeiro, J. M. Fernandes, J. C. CamposM., and D. Harrison. 2010. "The APEX framework: prototyping of ubiquitous environments based on Petri nets." *Human-Centred Software Engineering, volume 6409 of Lecture Notes in Computer Science* 6-21.

Thevenin, David, and Joelle Coutaz. 1999. "Plasticity of User Interfaces: Framework and Research Agenda." Grenoble: IOS Press.

W3C. 2011. "Accessible Rich Internet Applications (WAI-ARIA)." *W3C Candidate Recomendation.* January 18. URL: http://www.w3.org/TR/wai-aria/.

W3C. 2013. "CSS Values and Units Module Level 3." *W3C Candidate Recommendation.* July 30. URL: http://www.w3.org/TR/css3-values/.

W3C. 2013. "The picture element." *W3C Working Draft.* February 26. URL: http://www.w3.org/TR/html-picture-element/.

Wroblewski, Luke. 2011. *Mobile First.* New York: A Book Apart.

Wroblewski, Luke. 2012. "Multi-Device Layout Patterns." *Lukew* URL: http://www.lukew.com/ff/entry.asp?1514.

Yahoo! 2011. "Mobile Shooping Framework: The role of mobile devices is the shopping process." *Yahoo! Advertising Solutions.*