# Web Design Responsivo

Páginas adaptáveis para todos os dispositivos

Casa do Código

TÁRCIO ZEMEL

# Prefácio

A era pós-PC da computação pessoal chegou com tudo. Nos tablets e smartphones, a Web, finalmente, tem a chance de se tornar verdadeiramente ubíqua e onipresente. Ainda estamos no começo dessa nova fase da tecnologia, mas o futuro é inegavelmente da Web móvel e dos mais diversos aparelhos de navegação.

Essa nova perspectiva traz desafios importantes. Até ontem, Web significava um navegador instalado num Desktop controlado por mouse e teclado numa tela de tamanho confortável. Nossos sites foram construídos pensando nesse cenário. Mas, agora, temos telas pequenas, touch screen, redes móveis e muitas outras diferenças. Precisamos de uma nova Web. Uma Web que suporte essa explosão de dispositivos e, mais ainda, esteja preparada para o futuro de dispositivos que ainda nem conseguimos antever.

O Web Design Responsivo é a chave para essa nova Web. É pensar em páginas que se adaptem a todo tipo de dispositivo e contexto de uso. É sair das limitações de um browser Desktop e seu tamanho previsível, e pensar em páginas com flexibilidade que suportem todo tamanho de tela, qualquer tipo de resolução, interfaces com touch ou mouse. Pensar responsivamente é repensar a Web para o futuro.

E este livro é a ferramenta que você precisa pra entender os desafios do Web Design Responsivo. O Tárcio mostra os pilares de uma interface responsiva, explora os aspectos técnicos do HTML e CSS, e mostra ideias para uma interface móvel usável. É um livro que recomendo para todo desenvolvedor Web que queira participar ativamente da nova Web. Espero que você aproveite a leitura como eu aproveitei.

*Sérgio Lopes - @sergio_caelum*

*Instrutor e desenvolvedor na Caelum*

# Sumário

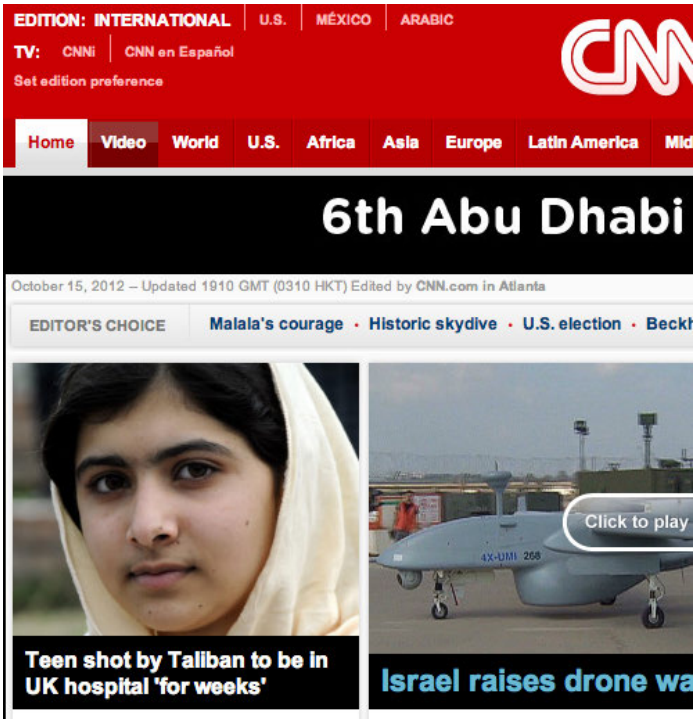
Capítulo 1

# Introdução

É possível desenvolver uma só apresentação para um site, um só *web design*. Mais além: que esse design seja bem apresentado em quaisquer dispositivos e, conforme se tenha planejado, que se adapte aos diferentes meios em que este site é acessado. Sim, é possível, mas não é o que vemos por aí.

Repare no site da CNN, em 2012, em um *browser*:


E agora o mesmo site, quando visto em um dispositivo que não um computador:



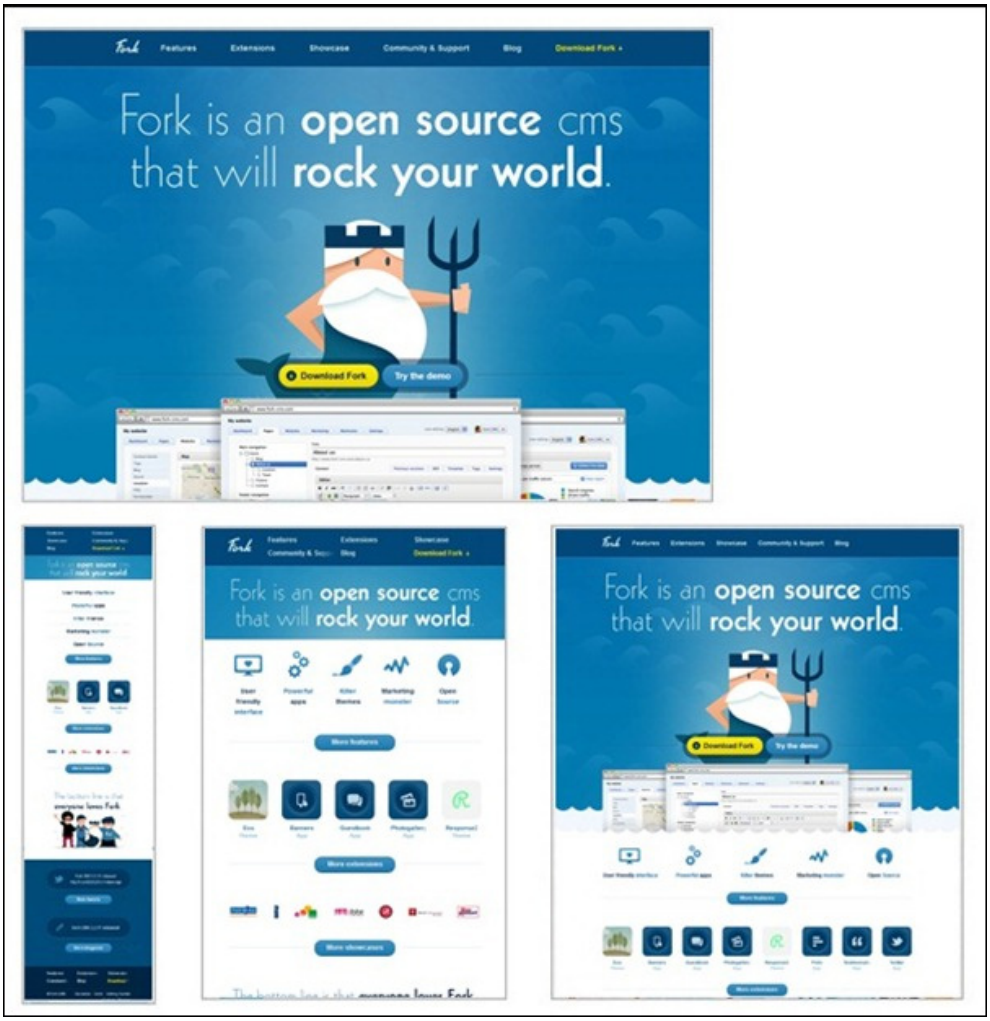
A página no dispositivo móvel aparece cortada e para ver a parte que está faltando é necessário fazer o *scroll* horizontal. Já na versão que é aberta em um *Desktop*, o site abre completo. A página da CNN não se adaptou aos diferentes meios pelos quais ele é acessado, já que, na versão móvel, nós somos obrigados a entrar em ação para visualizar o conteúdo relevante.

Repare o mesmo efeito no site da Editora Casa do Código:

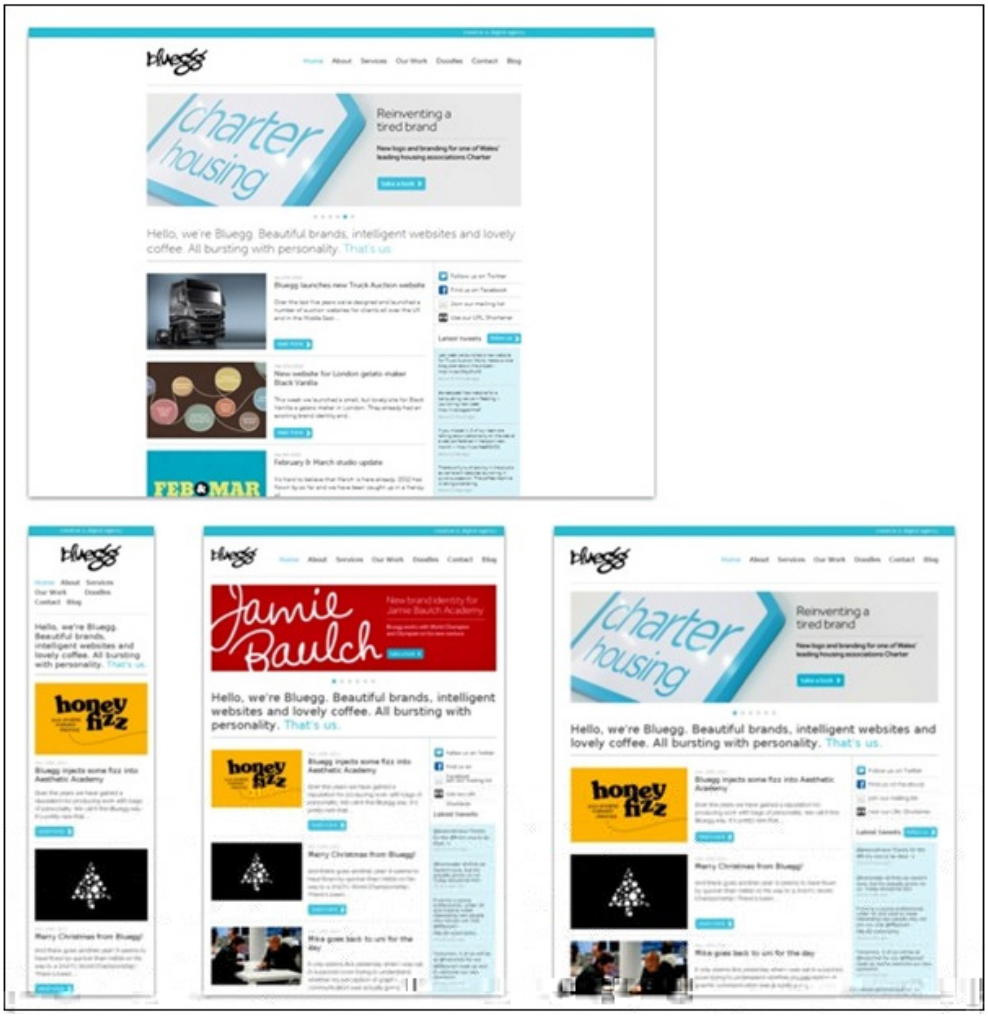

E o mesmo site, em dispositivo móvel pode ser visualizado da seguinte forma:

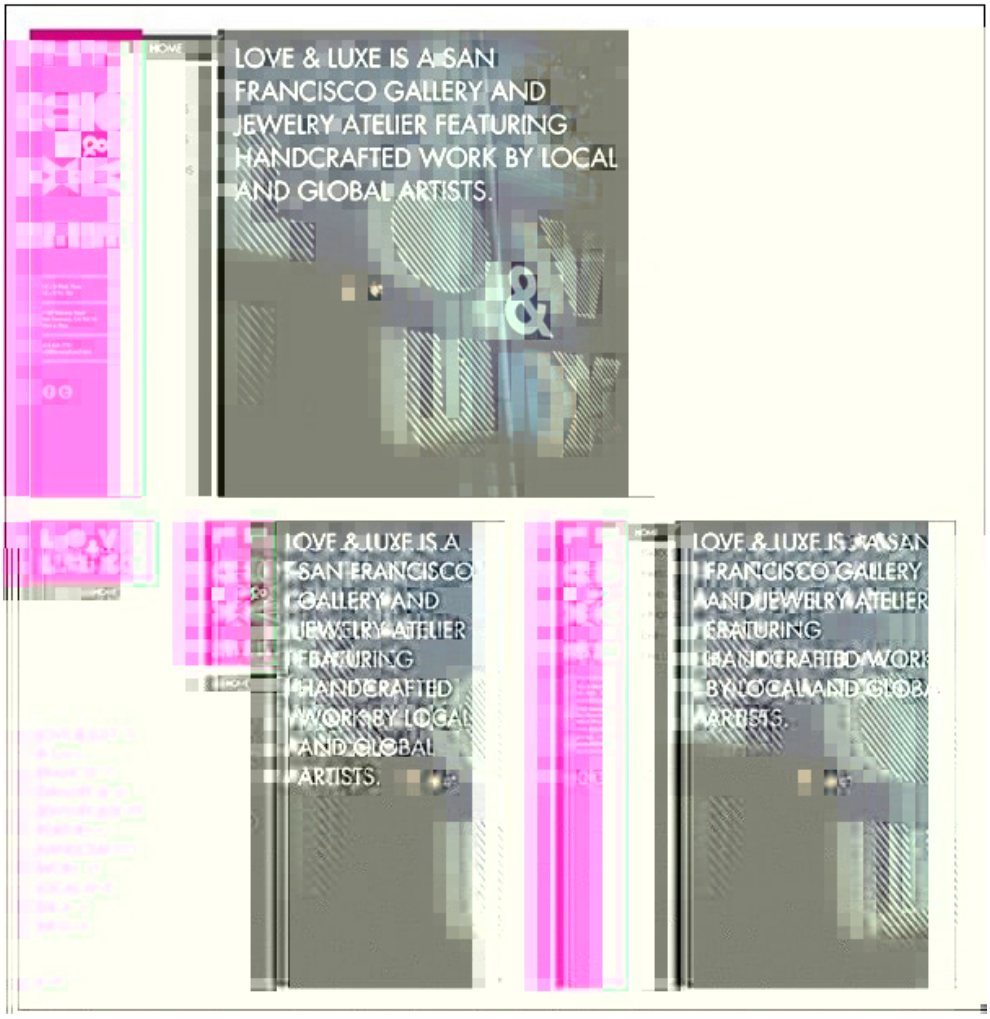

Para uma imagem mais impactante, veja o site do livro de arquitetura Java, do pessoal da Caelum, em diversos dispositivos:

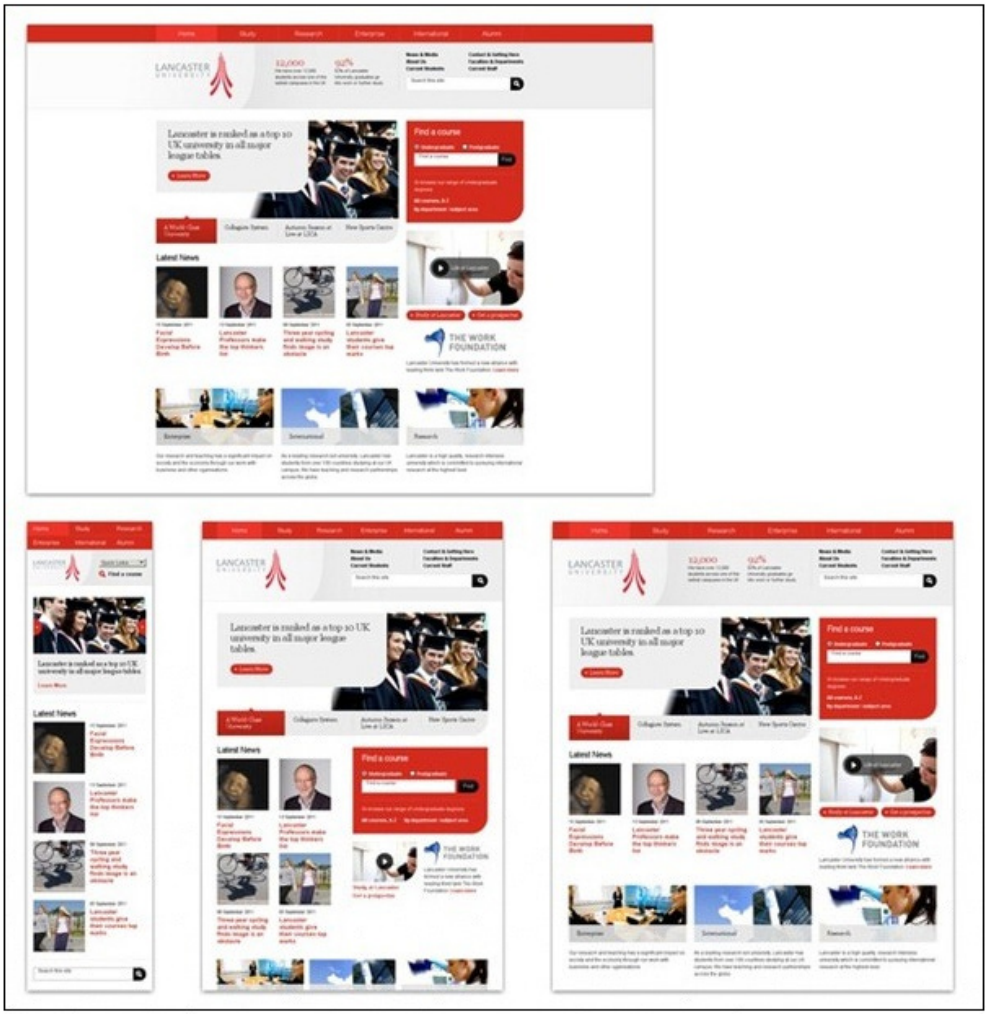
Veja outros exemplos incríveis de sites com essas características, que a revista TripWire reuniu:




Established Nottingham 2005
THE CELEBRATED NEW MISCELLANY OF
MR. SIMON COLLISON
A.K.A COLLY
POTTED AUTOBIOGRAPHY
THE SPLENDID JOURNAL
EXHAUSTIVE ARCHIVES
RECEIVING YOUR EMAILS
EXTERNAL REFERENCES [ VIEW ALL ]
LANYRD PROFILE
NEW ADVENTURES
FLICKR PHOTOGRAPHS
FOLLOW ME ON TWITTER
LAST.FM SCROBBLES
BLOODY FACEBOOK
DRIBBBLE SHOTS
INSTAGRAM PHOTOS
WRY & DERANGED


Bluegg
Home About Services Our Work Doodles Contact Blog
charter housing
Reinventing a tired brand
Hello, we're Bluegg. Beautiful brands, intelligent websites and lovely coffee. All bursting with personality. That's us.
Bluegg launches new Truck Auction website
New website for London gelatic maker Black Vanilla
February & March studio update
FEB MAR
Jamie Baulch
New brand identity for Jamie Baulch Academy
honey fizz
Bluegg injects some fizz into Aesthetic Academy
Merry Christmas from Bluegg!
Wise goes back to uni for the day


HOME
LOVE & LUXE IS A SAN FRANCISCO GALLERY AND JEWELRY ATELIER FEATURING HANDCRAFTED WORK BY LOCAL AND GLOBAL ARTISTS.
LOVE & LUXE IS A SAN FRANCISCO GALLERY AND JEWELRY ATELIER FEATURING HANDCRAFTED WORK BY LOCAL AND GLOBAL ARTISTS.
LOVE & LUXE IS A SAN FRANCISCO GALLERY AND JEWELRY ATELIER FEATURING HANDCRAFTED WORK BY LOCAL AND GLOBAL ARTISTS.



Sites com estas características possuem **web design responsivo**.

Um site com web design responsivo - ou *responsive web design* - pode ser acessado de um PC, notebook, smartphone, tablet, TV, geladeira, banheira - sim, realmente existem geladeiras e banheiras que acessam a internet! -, em suma, de qualquer dispositivo com acesso à rede, independentemente de sua resolução, de sua capacidade de cores, se é *touch* ou não. E, mesmo com essas diferenças dos dispositivos que podem acessar seu site, ele continua bem apresentado, inclusive com possibilidade de se alterar a ordem em que os conteúdos aparecem e, até mesmo, se determinados conteúdos serão ou não mostrados logo de cara!

Diga não a mais versões específicas para celulares; basta de linguagens próprias para mobile; chega de subdomínios ou diretórios específicos para atender ao "*público móvel*"! Não existe uma "*web mobile*" (*mobile web*). **A web é única**. Então, na minha opinião - e na de milhares de outros desenvolvedores web pelo mundo -, o web design responsivo é a resposta única para uma web única!

Não se trata de uma "moda" ou um hype da internet; não se trata de algo que chegou, vai angariar alguns fãs e sumir na próxima estação. O web design responsivo é uma nova forma de **pensar a web** e, dentro de pouco tempo, será tão vital e importante aos desenvolvedores e à experiência do usuário quanto o próprio HTML ou o CSS.

Ao terminar de ler este livro e fixar seus conteúdos, você fará parte do rol de profissionais de web capazes de desenvolver sites **responsivos** e poderá gritar, a plenos pulmões, que você, também, contribui para uma web e para um mundo melhor!

## 1.1 Estatísticas do mundo mobile

E não é somente porque usar essa tecnologia é algo divertido de se fazer. Existe uma forte base estatística que justifica por que devemos prestar muito atenção - e atuar! - no mercado *mobile*.

Por exemplo, havia uma previsão de que as vendas de celulares ultrapassariam as vendas combinadas de desktops e notebooks em 2012. Bem, a previsão estava errada: isso aconteceu em 2010 (http://ow.ly/awhu3), mesmo ano em que o número de celulares e *smartphones* bateu a casa dos **200 milhões** no Brasil! E mais: caso não aconteça antes, a previsão é de que o acesso *mobile* à internet ultrapasse o via PCs e notebooks em 2013 (http://ow.ly/awioc)!

É interessante notar que, conforme é destacado num artigo muito interessante sobre o mercado mobile do Brasil (http://ow.ly/bg7Wp), em 2011, a venda de *smartphones* cresceu **179%** no Brasil, com destaque para as informações de que:

- 33% das pessoas já tinham *smartphones*;
- 20% com funções consideradas avançadas, como Wifi, GPS etc;
- 41% acessava a internet;
- No último quadrimestre, o número de usuários *mobile* cresceu 40%.

Pode parecer incrível, mas se compararmos as vendas de aparelhos móveis com a taxa de natalidade, os dispositivos *mobile* têm uma taxa de crescimento **4 vezes maior que a da população mundial**!


E, partindo para o lado financeiro (http://ow.ly/awinf):

- O PayPal movimenta US$ 10 milhões por dia em pagamentos mobile;
- As vendas mundiais através de dispositivos móveis no eBay chegaram perto de US$ 2 bilhões em 2010 (são feitos cerca de 94 lances *por minuto*);
- Em dezembro de 2010, o número de usuários do aplicativo do Yelp estava na casa dos 3,2 milhões e 35% das buscas feitas no site são através desse app.

Levando em conta como é feito o uso do *mobile* pelas pessoas, também há alguns números recolhidos pelo Google em solo estadunidense que merecem atenção (http://ow.ly/awkw7):

- 79% usam *smartphone* como auxílio na hora de fazer compras (70% dentro da loja);

- 54% para procurar o endereço de uma loja;
- 49% para comparar preços;
- 44% para ler *reviews* de produtos;
- 74% tomam a decisão da compra baseados em informações obtidas no *smartphone*;
- 35% dos que pesquisam no *smartphone* compram o produto através dele;
- 88% que encontram informações no *smartphone* tomam a iniciativa no mesmo dia;
- 71% dos que buscam algo em um *smartphone* o fazem após verem um anúncio;
- 79% dos anunciantes ainda não têm um site otimizado para *mobile* (!).

### "Nosso Planeta Mobile: Brasil"


Em maio de 2012, a **Google** liberou, no mundo todo, uma pesquisa chamada de "Nosso Planeta Mobile". Felizmente, o Brasil também foi contemplado e um documento foi disponibilizado com os resultados obtidos. É altamente aconselhável que você tenha acesso e estude o relatório "**Nosso Planeta Mobile: Brasil - Como entender o usuário de celular**" (http://ow.ly/bg93Q). Já adiantando alguns números importantes:

- A difusão dos *smartphones* atinge 14% da população e esses proprietários de *smartphones* dependem cada vez mais de seus dispositivos. 73% acessam a internet todos os dias no *smartphone* e muitos nunca saem de casa sem ele;
- 88% dos usuários de *smartphones* procuram informações locais em seus telefones e 92% tomam decisões em decorrência disso, como fazer uma compra ou entrar em contato com a empresa;
- 31% dos usuários de smartphones fizeram uma compra pelo celular;
- Anúncios para celular são vistos por 94% dos usuários de *smartphones*;
- 75% dos usuários realizaram uma pesquisa em seus smartphones depois de visualizar um anúncio off-line.

## 1.2 Uma questão de conceito

Como vimos, já é uma realidade que os mais diversos dispositivos *mobile* chegaram para ficar, estão assumindo seu lugar (e o de outros) e, daqui a não muito tempo, prometem ser o padrão para acesso a web! Já é hora de começar a olhar para o desktop com um olhar saudosista, agradecendo por tudo o que ele fez por todos nós até hoje, mas, ao mesmo tempo, já se preparando para o que o futuro (breve) trará.

Em nosso dia-a-dia, já é possível perceber isso. Note que, nos lugares mais inusitados, as pessoas estão andando não somente com seus *smartphones* ligados e em pleno uso, mas, também, com seu *tablets*, *e-Readers* e muitos outros dispositivos com nomes extravagantes. Mas não pense somente em dispositivos com *displays* menores do que os tamanhos de monitor convencionais. Ou ainda não ouviu falar nas *Smart TVs* e nem viu alguém acessando sites através de um videogame?

Em quaisquer dos casos citados, a situação ideal é que os sites desenvolvidos sejam bem apresentados (e, até, *otimizados*) conforme o *device* que o está acessando no momento. Será que as informações exibidas para quem acessa um site de jogos através de seu videogame devem ser apresentadas exatamente da mesma maneira de quem realiza esse acesso usando um celular ou *tablet*? Obviamente que não!

Mas é aí que está o grande problema do desenvolvimento web "tradicional": os sites não são idealizados, desde sua concepção, para serem flexíveis e mostrados de forma adequada, seja lá por que meio de acesso a pessoa os estejam acessando. Perceba que, mais do que uma questão técnica, estamos tratando de algo **conceitual**!

E para começar a pensar em web design responsivo, o primeiro passo é começar a mudar seus conceitos!

### Qual a solução?

A solução (ou soluções) para que a web possa evoluir e os desenvolvedores possam atualizar seu *know-how* teórico e prático é, justamente, o que é tratado neste livro. Através dos conteúdos aqui apresentados, é que os profissionais de web tomarão conhecimento sobre técnicas como **layout □uído**, **imagens □exíveis**, **Media Queries** e muitas outras dicas úteis sobre o desenvolvimento de **web sites responsivos**!

Preparado?

## 1.3 Mas meu site já está bom para Android e iPhone!

Você pode ter um site focado para cada dispositivo, mas há diversas desvantagens nessa abordagem. Em especial, e quando houver um novo dispositivo móvel, com uma tela bem diferente? E quando precisar fazer uma alteração, terá de tomar cuidado com cada uma das várias versões do seu site.

Tomemos como exemplo o Wikipédia que, atualmente, vai redirecioná-lo para a página http://en.m.wikipedia.org/ se o acesso for *mobile*. Chamamos isto de estratégia do "m ponto" (`m.website.com`). Se você abri-la no desktop, a página flui e ocupa a tela toda. Fica feio e meio ruim de usar, mas é aceitável. Eles não usam *media queries* (que ainda veremos neste Livro) e nem nada mais avançado para *responsivar*. Mas é responsivo? Ninguém sabe dizer ao certo, fica numa área cinza...

Um outro conceito um pouco diferente é o de **One Web**. Que devemos ter uma URL só pra tudo, um sistema só e servir mesmo conteúdo pra todo mundo. O Design Responsivo costuma ser uma forma de implementar One Web com usabilidade adequada pra todo mundo.

Há ainda quem fale de uma outra técnica chamada **RESS - Responsive design + Server Side components**. Você serve a mesma URL, mesma página, mas ajusta no *server-side* algumas coisas sabendo qual *browser* é. Faz-se *user-agent sni ng*, que é menos preciso, mas ajuda em algumas coisas, como decidir se um link para ligação direta no celular deve ser exibido.

Você pode usar *server-side* pra manter a mesma URL (One Web), mas servir páginas específicas (não responsivas). Ou seja, há combinações diversas dessas técnicas (oneweb+responsivo, oneweb não responsivo, dois sites e responsivo, etc).

Nesse livro, focaremos em Design Responsivo com One Web. Ou seja, exploraremos os exemplos e cenários de sites únicos se adaptando através Media Queries, CSS flexível e recursos flexíveis. Esse cenário ninguém questiona se é responsivo ou não.

## 1.4 Para quem é este livro?

**Web design responsivo** é um assunto que, principalmente no Brasil, ainda é novidade. Até os mais veteranos (e, talvez, principalmente eles) desconheçam as técnicas e/ou, mesmo, seu conceito - gostaria de acreditar que não é o caso da maioria dos desenvolvedores, mas, caso seja, temos um trabalho de conscientização ainda maior a fazer! Por isso, querendo ou não, este livro destina-se a quem já possui alguns conhecimentos específicos na área de desenvolvimento web.

Este livro é destinado a todo e qualquer profissional, estudante ou aspirante a desenvolvedor web que deseja começar ou aprofundar seus estudos em web design responsivo. Através dos tópicos, tentamos encontrar a melhor maneira de expor este assunto - que, na verdade, não é tãão complexo, assim - da forma que, para nós, foi a cronologicamente adequada e didaticamente satisfatória.

Portanto, caso queira ter seu primeiro contato com design responsivo ou, caso já o tenha tido, mas, por algum motivo, não tenha entendido muito bem como as coisas funcionam, relaxe, prossiga com a leitura e tenha a certeza que, ao final do processo, você será mais um conhecedor dos mistérios profundos do web design responsivo!

## 1.5 Para aproveitar melhor este livro

Para entender e absorver bem o conteúdo, é bom que você já esteja um pouco familiarizado com HTML e CSS. Saber um pouco de JavaScript é desejável, não para acompanhar os conteúdos principais e mais importantes do livro, mas sim para acompanhar o conteúdo complementar, dicas extras, o *plus* do design responsivo, que muitas vezes precisa de um js para ser feito.

Mas fique tranquilo! Não é na maioria das técnicas que isso é preciso e, quando o for, tratam-se de materiais não-essenciais, sem os quais você ainda conseguirá entender como funciona o design responsivo para web.

Então, se você já é profissional do ramo e já passou (ou se encontra ou se interessa) pela área de front-end, não terá maiores problemas ao continuar com a leitura.

Se você é do back-end e não se interessa tanto pelo que acontece "lá na frente", acredito que, ainda assim, poderá aproveitar os conteúdos, dado que, geralmente,

mesmo o profissional não sendo exclusivamente alocado a uma área de atuação, conhece, pelo menos, as "nuances" do trabalho dos outros colegas de equipe.

Caso seja você um praticante-militante de web design, profissional de usabilidade, UX e afins, então, ou você dá uma lidinha em alguns tutoriais sobre HTML/CSS (caso já não saiba alguma coisa) ou, fique sabendo desde já, seu aproveitamento maior será sobre como planejar um site, desde sua concepção, para ser responsivo - principalmente, na filosofia *mobile first*, que será apresentada -, já que, desde que seja preciso ou acordado na equipe que "tal" ou "qual" projeto contará com web design responsivo, será obrigação **sua** (ou da equipe a qual você faça parte) pensar e apresentar as soluções visuais para os diferentes "breakpoints" do design.

Quer dizer, não importa, tanto, em que área dentro da "linha de produção" de desenvolvimento web você se encontre no momento. O que deve ficar claro é que você precisa conhecer a marcação e sintaxe do HTML, sabendo quando aplicar e para que servem as diferentes tags da linguagem; deve conhecer CSS, seus seletores e as respectivas propriedades e possíveis valores e; por último (e, no caso, menos importante), um pouco de JavaScript, caso queira se aprofundar um pouco mais em design responsivo e aprender algumas técnicas interessantes, também mostradas e explicadas no livro.

Capítulo 2

# Princípios de um web design responsivo

Já chegamos a comentar na Introdução o que é um web design responsivo, que é aquele web design que *responde* a quaisquer dispositivos/resoluções e, devido a uma série de características técnicas bem específicas, é bem apresentado em qualquer um deles.

Vamos nos aprofundar um pouco mais em suas origens, raízes e inspirações, e em como aplicá-lo.

## 2.1 Como surgiu o web design responsivo

Existe um site muito, muito bom e tradicional na web, que é o **A List Apart**: http://www.alistapart.com/

Ele existe desde 1998 e, com o passar dos anos, nunca deixou de brindar seus leitores com artigos e livros de excelente qualidade.

Um dos escritores do A List Apart é **Ethan Marcotte** que, em meados de 2010, publicou um artigo intitulado *Responsive Web Design* que mudaria, para sempre, a forma como se faz design para web.

http://www.alistapart.com/articles/responsive-web-design/

Numa tradução livre, este artigo começa assim:

*"O controle que os designers têm no meio impresso e, muitas vezes, desejam ter no meio web, é simplesmente um re exo da limitação da página impressa. Devemos aceitar o fato de que a web não tem as mesmas restrições e projetar (o web design) para essa exibilidade"*

E, no decorrer do artigo, Ethan explica seus conceitos e sugestões (usando tecnologia que já era existente à época de sua publicação) para que as páginas fossem projetadas usando o que ele chamou de **web design responsivo**. Até um site de exemplo foi mostrado no artigo (http://ow.ly/ate9K), o que deixou a comunidade de desenvolvedores em polvorosa!

Inspirado por conceitos de arquitetura e filosofia, a proposta de Ethan com a publicação daquele artigo foi mostrar um conjunto de técnicas que garantem responsividade a um web design.

Devido ao enorme feedback positivo e à ânsia da comunidade por maiores explicações e pormenores sobre o assunto, no ano seguinte, foi lançado o livro que, por razões óbvias, levou o mesmo nome do artigo que revolucionara a maneira de se fazer design na web: o **Responsive Web Design**.

E, como não poderia deixar de ser, muito da "inspiração" deste livro veio do que foi compartilhado por Ethan em seus escritos. Portanto, nada mais natural do que prestar nossa homenagem ao homem que liberou para o mundo energias de conhecimento web profundo, baseando vários dos conteúdos que aqui constam em seus ensinamentos.

*ank you, Ethan!*

## 2.2 A trinca tecnológica do web design responsivo

Para conseguir desenvolver um design responsivo para a web, 3 tecnologias principais (ou modos de aplicar essas tecnologias, se preferir) estão envolvidas:

1) Layout fluído;

2) Imagens e recursos flexíveis; e

3) Media Queries.

Desenvolver sites com **layouts □uidos** ( *uent layout*, também chamados de *grids exíveis*), ou seja, desde a concepção do projeto, primar pela não especificação de medidas fixas no layout do projeto, torna possível que haja uma adaptação "natural" e automática do que se apresenta na tela. Portanto, seja qual for a resolução do dispositivo que fez o acesso, evitamos barras de rolagem inconvenientes e/ou conteúdos "cortados", não exibidos em sua completude.

Mas de nada adiantaria se o conteúdo se adaptasse às mais diferentes resoluções se as imagens (e outros recursos) do site também não se adaptassem e não fossem **□exíveis**, não é verdade? Então, através de técnicas variadas, é possível fazer com que os *assets* (recursos como imagens, vídeos, etc) do site também sejam flexíveis para garantir que a experiência do visitante prime pela excelência, independentemente do dispositivo que esteja usando.

Para as mais diferentes resoluções dos aparelhos, as necessidades e desejos de uso do site podem se alterar. Afinal, uma pessoa não estaria errada ao imaginar que alguém que acessa um site através de um celular, cuja tela não tenha uma boa resolução, não precisa de uma *sidebar* (menu na lateral). O que esse usuário provavelmente está procurando são os conteúdos principais; então, nada mais justo que não onerar a visita com a exibição de uma barra lateral. As opções presentes na *sidebar* poderiam, por exemplo, serem apresentadas no fim de todo o conteúdo.

É com as **Media Queries** que é possível ocultar, fazer aparecer e reposicionar elementos e interações conforme a resolução atual que esteja sendo usada no momento da visitação. Afinal, o site não precisa (e, na verdade, não deve) ter, exatamente, a mesma aparência e disposição de seus elementos em qualquer resolução. Um iPad, um celular Nokia e um monitor de 27 polegadas possuem espaços, resoluções e necessidades bem diferentes.

Sabendo usar essa *trinca tecnológica* é possível criar designs responsivos para a web. É objeto deste livro fazer a apresentação e respectivas explicações (e integrações) de cada uma dessas principais tecnologias. Mas, antes, é interessante ter uma noção de alguns problemas comuns que trouxeram os desenvolvedores web até este ponto e conhecer alguns fatos notáveis sobre dispositivos, a web, a relação entre estes.

## 2.3 Resoluções de tela

Como se pôde ver quando apresentamos alguma estatísticas do mundo *mobile* na seção 1.1, os números envolvendo esse novo marco tecnológico são surpreendentes!

E estes números mostram, também, que a tendência é que mais e mais dispositivos surjam, muitos a cada ano. Então, juntando os já existentes com os que são lançados a cada período programado de tempo, temos um bocado de *devices* para prestar atenção... Ou nem tanto?

De maneira óbvia, é humanamente impossível prestar atenção e dar suporte a todo e qualquer dispositivo mobile existente e que esteja por existir. Na verdade, nem é preciso que isso seja feito.

## O atual (e não e□ciente) paradigma de acesso multidispositivo

Sem essa ideia em mente para o desenvolvimento, mesmo antes de se começar a escrever o primeiro caractere de código, o que se faz é o seguinte: escolhe-se uma técnica, qualquer, para identificar qual é a resolução do dispositivo (ou o dispositivo, em si) e, a partir do resultado, carrega-se recursos condizentes (CSS, JavaScript, imagens, etc) à realidade daquele *device*. Algo que também não é difícil de acontecer: a pessoa é redirecionada para um site à parte, feito para uma resolução ou dispositivo específico - se preferir: um "site compatível"... Não é preciso escrever parágrafos infindáveis sobre os custos de produção e retrabalho envolvidos ao se usar esta abordagem, não é verdade? Sem contar que, muitas vezes, queríamos uma experiência parecida, e não algo completamente diferente do site "normal".

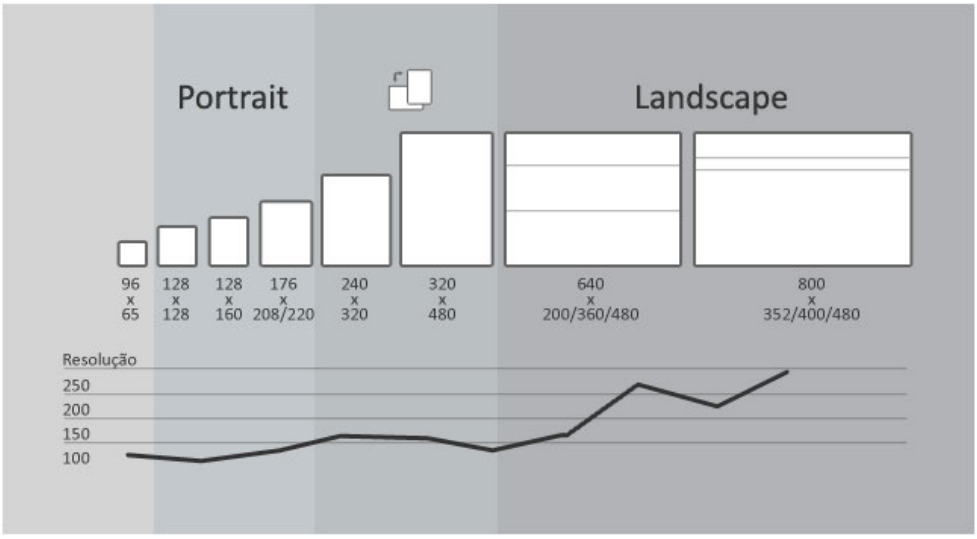
O equívoco comum é ainda não ter entendido que, na verdade, o que importa não é o tamanho físico da tela ou o dispositivo, em si, mas sua **resolução**! Tendo a atenção voltada às *resoluções*, é possível, com facilidade muito maior, desenvolver para uma gama incrível de dispositivos, sem se preocupar com, necessariamente, que dispositivos são esses ou o tamanho físico de suas telas.

Ou você quer ter que atualizar seus *scripts* de identificação de *devices* sempre que o mais novo *smartphone* do momento for lançado no próximo mês? Claro que não! E, certamente, qualquer profissional da área de desenvolvimento web, passando pelo estagiário de HTML ao gestor do negócio, também não.

Para se ter uma ideia, peguemos uma pesquisa feita com aparelhos vendidos entre 2005 e 2008, que contemplou análise de 400 *devices* diferentes (http://ow.ly/b72At), com intenção de levantar as principais resoluções de dispositivos usados para acesso à web, também levando em considerações as orientações desses dispositivos: *portrait* (retrato) e *landscape* (paisagem).

Constatações interessantes foram feitas, como, por exemplo, a diferença de tamanho entre o menor e o maior dispositivo que participaram da amostra, que chega

a ser de mais de **23 vezes** e o aumento considerável de *ppi* ("*Pixels per inch*", *densidade de pixels*) conforme o avanço das tecnologias.




Para ficar mais claro e trazer o entendimento com exemplos mais concretos, tomemos o exemplo de alguns dispositivos, especificamente, para se ter noção das diferenças de resolução que podem existir. Vejamos quais as resoluções específicas de alguns dos aparelhos comuns, que podem ser encontrados em qualquer lugar. Para facilitar, o gráfico disponibilizado no blog WebDev-il (http://ow.ly/bazAV), faz uma separação lógica dos *devices* por Sistema Operacional.

Perceba que, neste gráfico, existem 2 indicações para iPhone: uma é o retângulo que representa a resolução de 320x480 e o maior de todos, 640x960, também! A diferença é que a o iPhone 4 tem uma resolução bem superior. E o iPhone 5 aumentou ainda mais essa diversidade, com seus 640x1136 pixels.

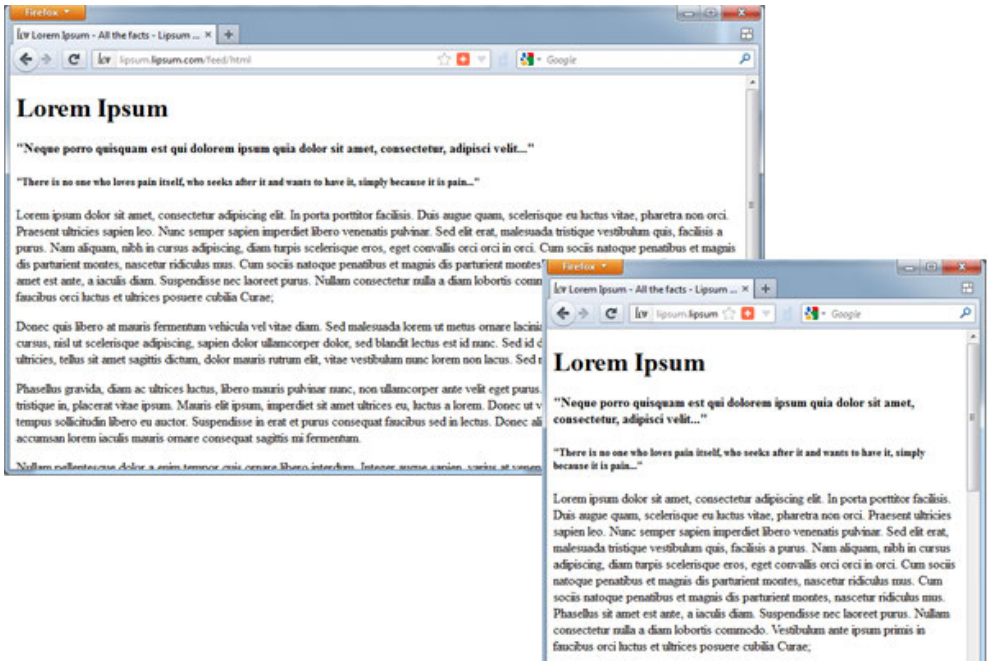
O que importa não é o tamanho físico da tela ou o dispositivo em si, mas sim sua **resolução**. Fixou?

Capítulo 3

# layout fluído

Um **layout □uído** ou, como originalmente chamado por Ethan Marcotte, **grid □exível**, é o primeiro passo para desvendar os mistérios do web design responsivo. Para se conseguir um layout fluído num projeto web, a principal medida a ser tomada é: **não usar medidas absolutas no CSS**!

Aliás, você mesmo, ao começar a codificar seus primeiros exemplos de HTML, já notou isso, mas aposto que esqueceu! Ou não se lembra de como aquele parágrafo cheio de "*Lorem ipsum...*" se adaptava a qualquer tamanho de janela do navegador? E por quê? Porque não havia nenhuma especificação de largura! Simples assim.




Portanto, ao especificar tamanhos, espaçamentos, margens, paddings, enfim, ao estipular qualquer medida referente ao layout das páginas do site, é preciso usar valores relativos, como **porcentagens** e **ems**, como veremos.

Antes de prosseguirmos, vamos ver quais são os tipos de medidas em CSS.

## 3.1 Tipos de medida em CSS

Embora existam outros tipos de medidas em CSS, vamos basear esta explicação nas 4 principais: **pixels**, **pontos**, **porcentagens** e **ems**.

- **Pixel (px).** É a unidade de medida fixa mais usada nas CSS. Comumente falando, um pixel é um ponto indivisível na tela de exibição de um dispositivo - embora, mais recentemente, tenhamos vários "tipos de pixel" (http://ow.ly/atmIu). Não raramente, web designers preferem usar este tipo de medida para fazer uma estrutura HTML/CSS em *pixel perfect*, não medindo esforços para estruturar seus documentos para que fiquem idênticos à imagem do web design.

- **Ponto (point).** Pontos são tradicionalmente utilizados para CSSs de impressão. Um ponto é igual a 1/72 polegadas. Assim como os pixels, pontos são unidades de tamanho fixo.

- **Ems (em).** O "*em*" é uma unidade escalável. Quando se trata do tamanho da fonte, 1em é igual ao tamanho atual da fonte do elemento-pai. Por exemplo, se o tamanho da fonte do elemento é 12pt, 1em é igual a 12pt. *Ems* são escaláveis por natureza. 2em seria igual a 24pt, 0.5 seria 6pt, etc.

- **Porcentagem (%).** A unidade porcento é muito parecida com a unidade "*em*", mas possui algumas diferenças fundamentais. Em primeiro lugar, o atual tamanho da fonte é igual a 100% (ou seja 12pt = 100%). Durante o uso da unidade porcento, o texto permanece totalmente escalável para dispositivos móveis.

Como foi visto, a diferença básica entre pontos, pixels, *ems* e porcentagens é que os dois primeiros são unidades de medida fixas e os dois últimos, variáveis. Eles são relativos, escaláveis e se adaptam e mantêm relações de tamanho com outros elementos de um documento - eles possuem um **contexto**.

Portanto, fica claro que usar pts ou pxs não é o adequado para designs responsivos (num primeiro momento, já que, com a prática e/ou experiência e/ou demanda do projeto, pode ser necessário especificar alguns valores fixos em determinadas situações), o que nos deixa com as 2 outras opções. Mas qual a diferença entre usar ems ou porcentagens?

## Ems x Porcento

Devido às características que foram analisadas previamente, parece que é a mesma coisa usar em ou porcentagem. Entretanto, existem algumas pequenas diferenças.

O importante é saber que, quando falamos em layout, marcar unidades com porcento fornece uma exibição mais consistente e acessível para os visitantes. Quando as configurações de exibição do cliente se alteram, as medidas marcadas com % se alteram de maneira mais razoável, permitindo que a legibilidade seja preservada, bem como a acessibilidade e o design visual.

Portanto, apesar de ser possível usar qualquer um dos tipo de medidas relativas, é meio que consenso entre os desenvolvedores - e, acredito, este "consenso" surgiu através de muitos testes e experiência com responsividade - **usar porcentagem para lidar com tamanhos no layout** (larguras, margens, espaçamentos, etc) e **usar ems**

**para lidar com fontes**. `em` pode até ser usado fora de textos, mas vai ser sempre uma **medida relativa ao** `font-size`; já o porcento é relativo ao `font-size` quando usada em `font-size`, mas, quando usada com outras medidas, é **relativa à largura do elemento-pai**.

## 3.2 A fórmula mágica do web design responsivo

Quando, desde sua concepção, um projeto tem por objetivo contar com design responsivo, é importante que os desenvolvedores envolvidos pensem de "forma relativa", ou seja, em implementar este layout se valendo de medidas relativas no CSS. Relembrando, deve-se pensar em atribuir as medidas do layout em **porcentagens** e o tamanho de fontes em **ems** ao invés de pixels.

Se, na "antiga maneira de pensar", houvesse, por exemplo, um título com tamanho de fonte de 24px, então, seria preciso converter essa medida em **em**. E, se você está prestando atenção nos conteúdos que nos trouxeram até este ponto, você deveria estar se perguntando: *"Mas como converter para em e continuar com a aparência de 24px?"*. E essa, meu caro colega de profissão, é, realmente, a pergunta certa a se fazer!

Existe uma **fórmula** para realizar esse cálculo. Uma forma simples e rápida de realizar a conversão de tamanhos absolutos para tamanhos relativos em layouts para a web. E não é nada com que você deva se preocupar ou sentir um arrependimento profundo ao imaginar que deveria ter prestado mais atenção às aulas de matemática da época do colégio. É uma fórmula tão simples que, por si só, contém a beleza e a elegância do design responsivo.

Se estivéssemos em algum ponto próximo da Idade Média, alguns diriam que essa é uma fórmula mágica! Mas, não, não é magia.

> **Fórmula mágica do web design responsivo**
>
> **Alvo / Contexto = Resultado**

- **Alvo.** Elemento-alvo com a medida atual;
- **Contexto.** Onde o elemento-alvo está (baseado no elemento-pai);
- **Resultado.** O valor relativo que se está procurando.

Com essa fórmula rápida, é possível realizar cálculos simples para saber o resultado de conversões de medidas absolutas do CSS para medidas relativas. E isso vale tanto para o cálculo de **tamanho de fontes** - existe uma certa convenção de que o tamanho padrão de fontes em *browsers desktop* é de 16px -, quanto para **medidas de layout**!

Então, voltando ao nosso exemplo, agora fica simples descobrir qual a equivalência em ems de 24px. Basta aplicarmos a fórmula.

**24 / 16 = 1,5**

Ou seja, ao pegar nosso alvo (o título) de `24px` e dividir pelo contexto (no caso, seu elemento-pai, que é `body`, possui tamanho fonte de 100%, ou, no caso, `16px`), temos como resultado 1,5. Como estamos tratando de tamanho relativo de fontes em CSS, o resultado final é `1.5em`.

Imaginemos que haja um link dentro do título, que, no planejamento, teria `11px` de tamanho. É preciso converter isso para medida relativa. Aplicando a fórmula obtemos `11 / 16 = 0,6875`, certo? **Errado!**

Já que o link se encontra dentro do título, **o contexto mudou!** Agora, o elemento-alvo não está mais no "contexto geral" da página (`body`). Então, para conseguir o resultado correto, a variável de contexto da fórmula muda.

**11 / 24 = 0,45833333333333**

O elemento que, em medidas absolutas, deveria ter `11px`, está dentro do contexto do título (que, em medidas absolutas, teria `24px`), que resulta em `0.45833333333333em`.

Ethan Marcotte sugere que, quando o resultado da divisão contiver tantos números depois da vírgula, para não ficarmos tentados a arredondar e usar um número "mais enxuto", deixando o resultado com tantos números quanto sua calculadora conseguiu exibir. No entanto, o Autor corta pra ficar com 4 números depois da vírgula e nunca teve problemas. Faça seus próprios testes e opte pelo que lhe trouxer melhores resultados.

## 3.3 Exemplo de um layout fixo

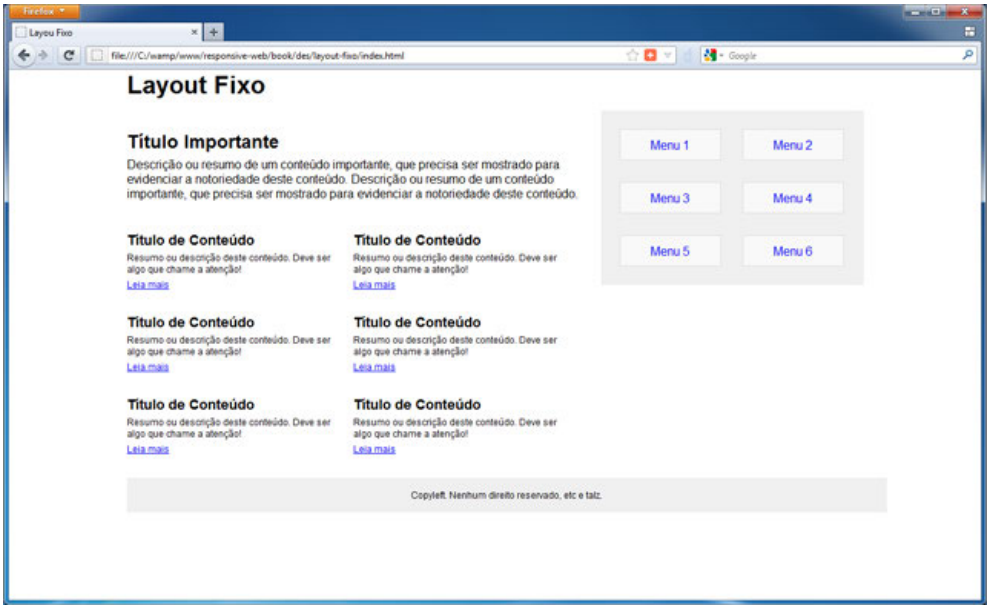
Como você já deve ter percebido a esta altura do campeonato, é possível aplicar nossa fórmula mágica tanto para descobrir a equivalência relativa de tamanhos de fontes, quanto para desenvolver um layout fluído. Para um melhor entendimento, vamos nos basear em um modelo de página fictícia, que ajudará bastante para aplicarmos os conceitos que estamos vendo.

### HTML de um layout □xo

Vejamos um exemplo do que encontramos no dia-a-dia da navegação na web: um site com **layout □xo**. A intenção é mostrar, através de um *layout* simples, a marcação de HTML e CSS que se encontra comumente, por aí.

> **HTML e CSS**
>
> O objetivo deste livro não é discutir teorias, técnicas ou condutas de escrita de HTML e CSS. Questões como espaçamentos, uso ou não de classes ou IDs, espaçamentos, identação, etc, fogem do escopo do livro.
>
> Se você quer conhecer mais, não deixe de conhecer o livro do Lucas Mazza, de HTML5 e CSS3, também pela Casa do Código.

Então, imaginemos uma página de um site com **layout □xo** que se pareça com isso:




Seu HTML é simples - como todo HTML deve(ria) ser - e não apresenta nada diferente do convencional. Provavelmente você já deve conhecer HTML, mas, para deixar bem claro sobre a função de nosso código, vamos a uma sucinta explicação.

Primeiramente, temos toda a estrutura da página "englobada" por um contêiner. Essa é uma técnica muito comum:

```
<div class="container">
    [...]
</div>
```

Existem 3 divisões principais, dentro do `div container`: um cabeçalho, o espaço para a inserção dos conteúdos principais e um rodapé. Novamente, sem nenhuma novidade para as técnicas mais básicas de estruturação em HTML:

```
<div class="container">
    <header class="header">
        [...]
    </header>

    <div class="content">
        [...]
    </div>

    <footer class="main-footer">
        [...]
    </footer>
</div>
```

Neste modelo de estrutura HTML, `header` e `footer` costumam sempre ter o mesmo conteúdo, independentemente da página do site que está sendo visitada no momento. Portanto, fica óbvio que os conteúdos principais do site ficam em `content`.

Dentro de `content`, também temos 2 "porções de conteúdo": a parte dos conteúdos principais, propriamente ditos, e uma *sidebar* - também conhecida como "barra lateral".

```
<div class="container">
    <div class="content">
        <div class="content-main">
            [...]
        </div>

        <aside class="content-sidebar">
            [...]
```

```
        </aside>
    </div>
</div>
```

Portanto, a estrutura deste site simples, demonstrada através desta página-exemplo, basicamente teria a mudança de conteúdos acontecendo em `content-main`. Nesta página, há uma chamada para o conteúdo mais recente ou mais importante em dado momento (a que chamamos, no caso, de `hero`), seguido de chamadas para outros conteúdos, cada uma mostrando seu respectivo título, um resumo/chamariz e um link para se acessar a página, na íntegra.

Como a estrutura foi idealizada para ser mantida e se repetir em todas as páginas do site, para os outros tipos de páginas, basta alterar os elementos HTML dentro de `content`.

No caso, essa página de exemplo foi estruturada com 6 destas "chamadas", mas a estrutura permite que sejam feitas quantas sejam precisas (preferencialmente, em quantidade par, para manter sua estrutura de apresentação).

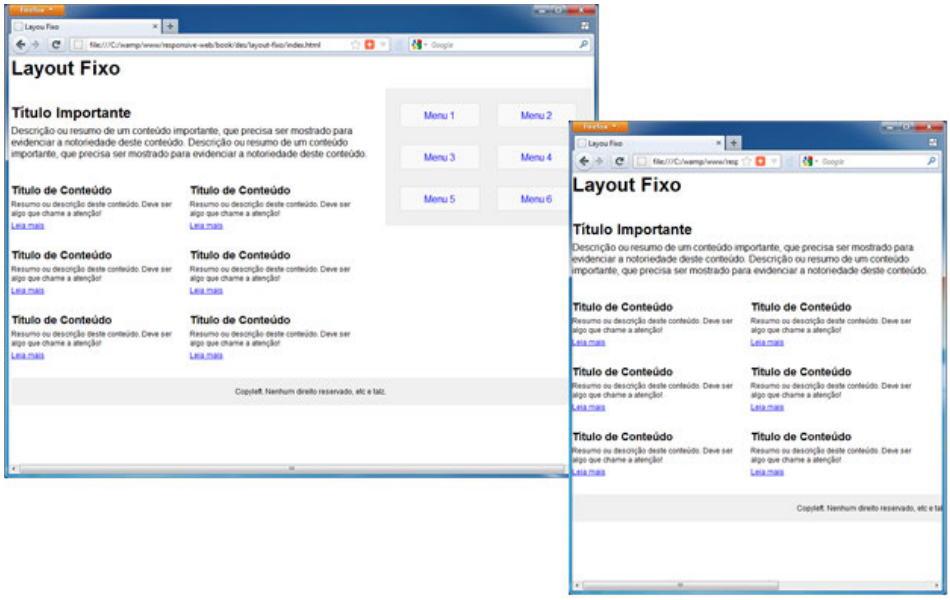
```
<div class="content">
    <div class="content-main">
        <article class="hero">
            <h2>
                Título Importante
            </h2>

            <p class="brief">
                Descrição ou resumo de um conteúdo
                importante, que precisa ser mostrado para evidenciar
                a notoriedade deste    conteúdo.
            </p>
        </article>

        <div class="last-contents">
            <article class="last-content-call">
                <h2 class="secondary-title">
                    Título de Conteúdo
                </h2>

                <p class="brief">
                    Resumo ou
                    descrição deste conteúdo.
```

```
                    Deve ser algo que chame a atenção!
                </p>

                <a href="#">Leia mais</a>
            </article>

            <article class="last-content-call">
                <h2 class="secondary-title">
                    Título de Conteúdo
                </h2>

                <p class="brief">
                    Resumo ou
                    descrição deste conteúdo.
                    Deve ser algo que chame a atenção!
                </p>

                <a href="#">Leia mais</a>
            </article>

            [...]
        </div>
    </div>

    <aside class="content-sidebar">
        <nav class="main-nav">
            <ul>
                <li><a href="#">Menu 1</a></li>
                <li><a href="#">Menu 2</a></li>
                <li><a href="#">Menu 3</a></li>
                <li><a href="#">Menu 4</a></li>
                <li><a href="#">Menu 5</a></li>
                <li><a href="#">Menu 6</a></li>
            </ul>
        </nav>
    </aside>
</div>
```

O código completo pode ser visto neste link: https://gist.github.com/3630605

Até então, tudo normal. Inclusive quando alguém redimensiona a janela - ou, o que é mais comum, acessa usando uma resolução menor do que a mostrada no

exemplo -, o conteúdo continua no mesmo lugar, eventuais barras de rolagem do navegador aparecem. Enfim, tudo normal e como o esperado:




Quando alguém acessa através de um dispositivo móvel, principalmente com os que temos atualmente, o próprio *device* se encarrega de fazer sua mágica para que todo o site seja apresentado na telinha como se aplicasse um *zoom out* nele todo. Com isso, a pessoa tem uma visão geral do site (por o estar vendo "pequenininho") e, caso algo seja do seu interesse, através de um botão pressionado ou gesto feito na tela do dispositivo, consegue dar um *zoom in* e ver melhor o que é de seu interesse no momento.

A partir do momento que a pessoa já viu o que queria, ou vai "deslizando" o site para encontrar outras coisas do interesse, ou dá um *zoom out* para ter novamente a "visão topográfica". Esse processo pode acontecer mais vezes, caso o usuário não consiga encontrar a informação com rapidez.

Essa facilidade que os dispositivos móveis nos oferecem realmente facilita o processo de navegação, já que nem todos os sites estão tecnicamente preparados para apresentar seus conteúdos nas mais diversas resolução possíveis. Sim, realmente é um processo que, depois de algum tempo, é cansativo, repetitivo e, concordemos, não atende aos padrões e gostos de usabilidade um pouco mais rigorosos.

O ideal é que o layout do site - e, logicamente, seu conteúdo - seja bem apresentado em todos os tipos de resolução. O desejável - e, muitas vezes e cada vez com mais frequência, o *esperado* - é que o site seja inteligente o bastante para apresentar o melhor conteúdo através da melhor maneira possível, não importando qual dispositivo ou qual resolução estejam sendo utilizados no momento do acesso.

## CSS de um layout □xo

Esse HTML de exemplo que mostramos deve ser acompanhado por um pouco de CSS, e é justamente isso que iremos abordar agora. Como você verá, é um CSS que não tem absolutamente nada demais, servindo para dar uma leve estilizada e apresentar nossa página de um layout fixo de uma maneira um pouco mais agradável e organizada.

Começaremos com a forma mais básica de CSS reset, servindo somente para ilustrar a necessidade de que haja um esforço inicial na criação das folhas de estilo de uma padronização entre os diferentes navegadores. Pessoalmente prefiro, ao invés de um reset, um "padronizador de CSS", tal como o CSS que podemos encontrar no HTML5 Boilerplate (http://html5boilerplate.com/); mas, para fins de exemplo, vamos trabalhar com um reset básico. Além disso, vamos colocar uma fonte comum para os elementos.

```
* {
margin: 0;
padding: 0;
}

html {
font-family: Arial, Helvetica, sans-serif;
}
```

Começando a mexer nos elementos especificados no HTML, vamos trabalhar com o `.container`, que é o *wrap* de toda a página. A ideia é termos um site centralizado, bastante comum nos dias de hoje. A largura "ideal" para tal modelo de página não existe, portanto, podemos arbitrar o que melhor nos aprouver.

```
.container {
margin: 0 auto;
width: 960px;
}
```

Aproveitando, vamos mexer, também, no título principal desta página, colocando um tamanho de fonte considerável para que não haja dúvidas quanto a sua importância.

```
h1 {
font-size: 32px;
}
```

Como teremos uma padronização para elementos do tipo `.brief`, não há mal algum em já estilizar este elemento "genérico" de nosso site-exemplo.

```
.brief {
margin: 5px 0;
}
```

Os conteúdos principais do site ficam em `.content` que, por sua vez, abriga `.content-main` e `.content-sidebar`, respectivamente (lembra-se do SEO?). E, dentro de ambos, mais estilos para que nosso exemplo fique numa estrutura interessante e funcional.

```
.content-main {
float: left;
width: 593px;
}

    .hero {
    margin: 25px 0;
    }

    .last-contents {
    font-size: 12px;
    }

        .last-content-call {
        float: left;
        margin: 15px 15px 15px 0;
        width: 280px;
        }

            .last-content-call .secondary-title {
            margin-bottom: 0;
```

```
            }

            .last-content-call .brief {
            margin: 5px;
            }

.content-sidebar {
background-color: #F0F0F0;
float: right;
padding: 10px;
width: 322px;
}

    .main-nav ul {
    list-style-type: none;
    }

        .main-nav li {
        background-color: #F9F9F9;
        float:left;
        margin: 15px;
        outline: 1px solid #DEDEDE;
        text-align: center;
        width: 130px;
        }

            .main-nav a {
            display: block;
            padding: 10px;
            text-decoration: none;
            }
```

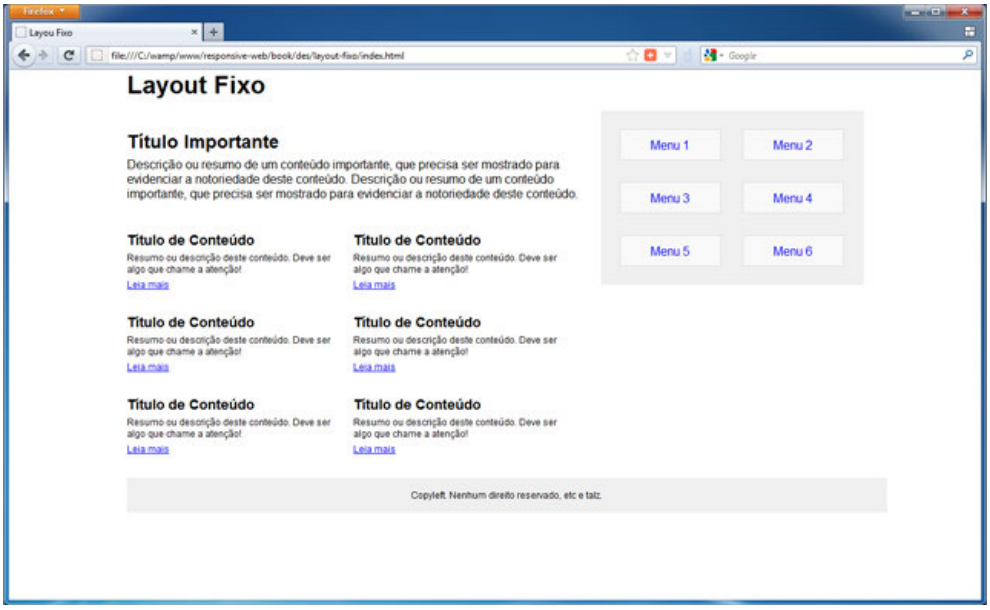
Por fim, temos nosso `.main-footer` que, no momento, serve somente para exibir algumas informações de *copyle* , mas nada impediria de conter um menu secundário, com links de menor importância que nosso menu principal, localizado na barra lateral.

```
.main-footer {
background-color: #F0F0F0;
clear: both;
float: left;
```

```
font-size: 12px;
margin: 15px 0;
padding: 15px;
text-align: center;
width: 100%;
}
```

Como prometido, é um CSS bastante simples, sem implementações polêmicas e/ou técnicas mirabolantes para conseguir efeitos diversos. O objetivo deste livro não é esse. Nosso CSS de um layout fixo completo pode ser encontrado em: https://gist.github.com/3630828

Com isso, terminamos nosso arquivo `style.css` que, como você bem observou no código completo do HTML, é referenciado e fará com que a camada de conteúdo receba sua devida estilização. Abrindo ambos os arquivos em um bom browser, você será premiado com uma tela muita parecida com:




Finalizamos o esqueleto de nossa página de exemplo. Não foi nada de mais, obviamente, mas, neste ponto, você pode estar se perguntando: "E agora, como transformar esse layout fixo em um layout fluído?". E essa é exatamente a pergunta certa a se fazer.

## 3.4 Metatag viewport

Veremos como converter um layout fixo em layout fluído. Primeiro, devemos nos ater a um dos mais importantes detalhes técnicos quando o assunto é web design responsivo: a **meta tag viewport**.

### Meta tags

Teoricamente, se você está lendo este livro, já deve saber o que é uma **meta tag** HTML. De qualquer forma, uma breve revisão pode ser útil.

O termo *meta tag* é, na verdade, formado por 2 palavrinhas que se juntam para dar seu significado. E, para entender este significado, é preciso entender as 2 palavras, em separado:

- **Meta** é um prefixo grego que significa algo como "após", "que ultrapassa", "que engloba", "que está além". Este não é um prefixo utilizado somente no desenvolvimento web; você já deve ter ouvido e lido palavras como "metabolismo" e "metamorfose". Este prefixo meta refere-se à "coisa sobre a própria coisa" ou, em outras palavras, dados sobre dados, informação sobre a informação.
- **Tag** é uma palavra do inglês que significa "etiqueta" ou "rótulo". Tanto nos EUA, no Brasil ou em qualquer parte do mundo, etiquetas servem para identificar, nomear e marcar algo ou alguma coisa para que possa ser identificado e/ou corretamente catalogado.

Então, juntando os 2 termos, "meta" e "tag", temos *meta tag*, que são *tags* que descrevem o documento web a qual pertencem ("informação sobre a informação"); as meta tags são para descrever informações sobre sites e páginas que as contém; informam sobre qual conteúdo está ali e mostra informações extras a respeito deste conteúdo.

As metatags são colocadas como elementos-filho de `head` no HTML. Sua sintaxe obedece à seguinte convenção:

```
<meta name="nome-da-meta" content="conteudo-da-meta">
```

Então, para qualquer das meta tags existentes que você for usar, a convenção para seu uso é esta, onde se informa, primeiramente, de qual meta se está inserindo e, posteriormente, qual valor se dá.

No documento do W3C, "Visual formatting model" (http://ow.ly/baEmW), é explicado que:

*"Agentes do usuário de mídia contínua em geral oferecem aos usuários uma viewport (uma janela ou outra área de visualização na tela) através do qual os usuários consultam um documento. Os agentes podem alterar o layout do documento quando o viewport é redimensionado. Quando a viewport é menor que a área da tela em que o documento é renderizado, o agente deve oferecer um mecanismo de rolagem"*

Ou seja, dentro do contexto de desenvolvimento web, *viewport* é a parte visível da página web que é renderizada pelos navegadores (desprezando painéis, barras de menu, plugins, barra de rolagem, etc).

### O que é a meta tag viewport

Os *browsers mobile* tentam exibir páginas web feitas somente para *desktop* (no momento, a imensa maioria de todas elas) ajustando, automaticamente, o *zoom* do *display*, e isso pode ser problemático para os sites que já foram planejados/otimizados para telas pequenas - todos os que você fará a partir da leitura deste livro, certo?

Felizmente, existe uma meta tag para contornar essa característica dos navegadores. É a *meta tag viewport*. Assim como qualquer outra meta tag, ela possui o formato:

```
<meta name="viewport" content="">
```

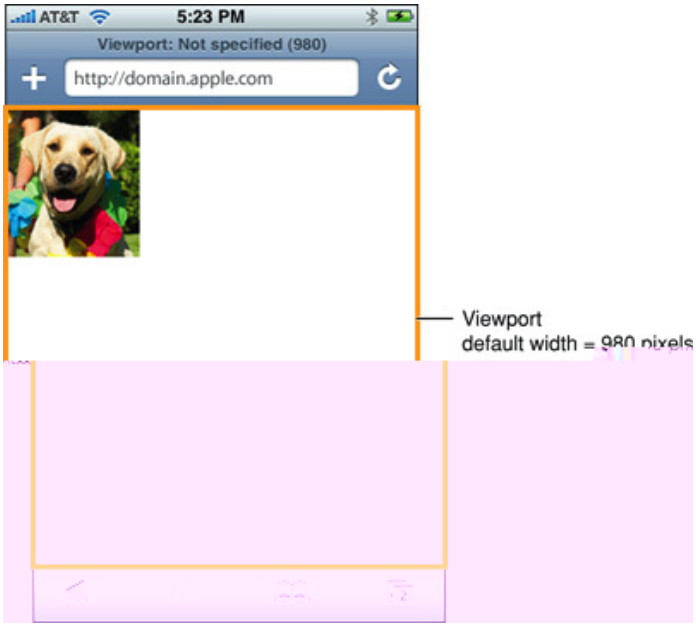
Sendo que, em `content`, é possível especificar uma diversidade de parâmetros e valores conforme o tipo de visualização que se configurar às páginas. Se preferir, pense que com a *meta tag viewport*, é possível apresentar "resoluções personalizadas" aos visitantes para determinados *devices*. Os principais e mais usados parâmetros de `content` são:

- **width**: define a largura da *viewport*;
- **height**: define a altura da *viewport*;
- **initial-scale**: define a escala inicial (*zoom*) inicial da *viewport*.

Sendo que é possível usar mais de um parâmetro, ou mesmo todos, como valor de `content` da *meta tag viewport*.

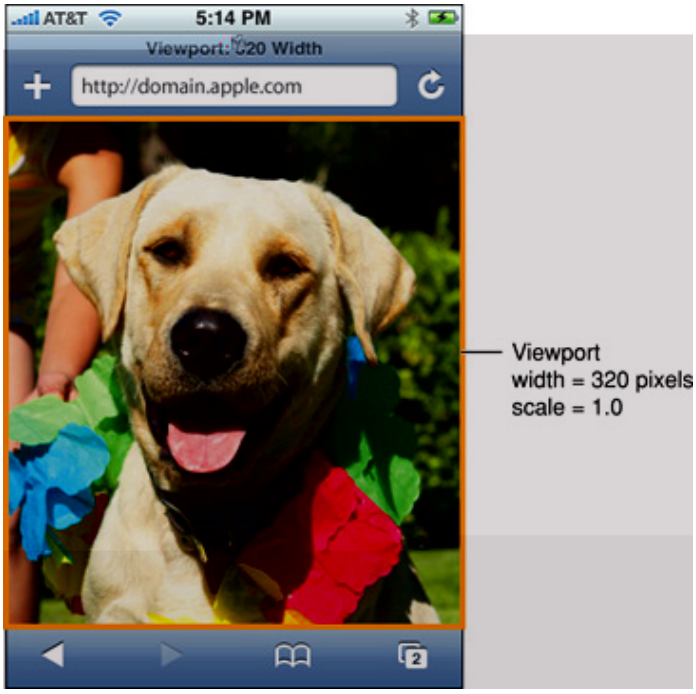
### Exemplos de uso da meta tag viewport

A própria Apple, que originalmente propôs e colocou em uso a *meta tag viewport*, apresenta alguns exemplos que elucidam bastante a questão (http://ow.ly/baGbr). Veja uma imagem de `320x356px`, apresentada num iPhone, renderizada usando as configurações *default* de *viewport*.



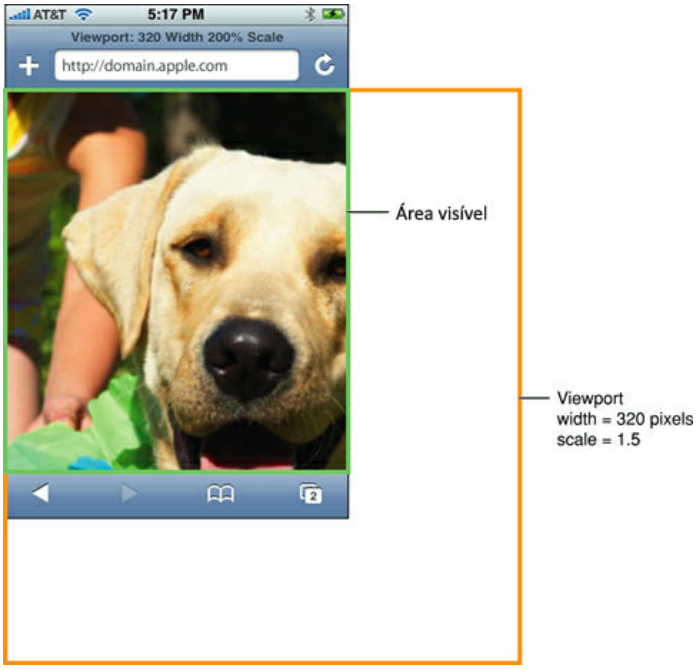

Agora, a página com o *zoom* na *viewport* sendo o mesmo do da própria imagem.

```
<meta name="viewport" content="width=320,initial-scale=1">
```



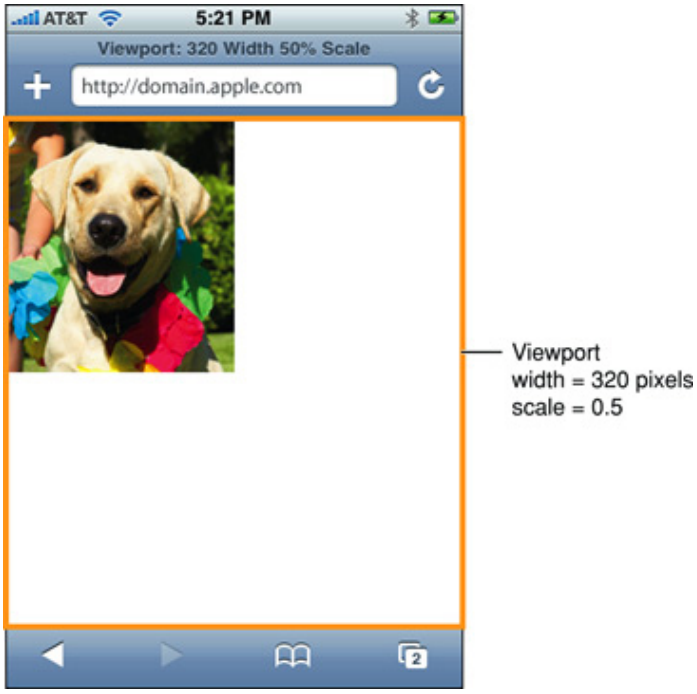

No entanto, como se trata de configurações que podem ser escolhidas conforme a necessidade do projeto, o *zoom* na *viewport* pode ser maior ou menor do que a área visível. Se o *zoom* é maior do que a área visível, então será possível deslizar a tela para ver mais da página.

```
<meta name="viewport" content="width=320,initial-scale=1.5">
```

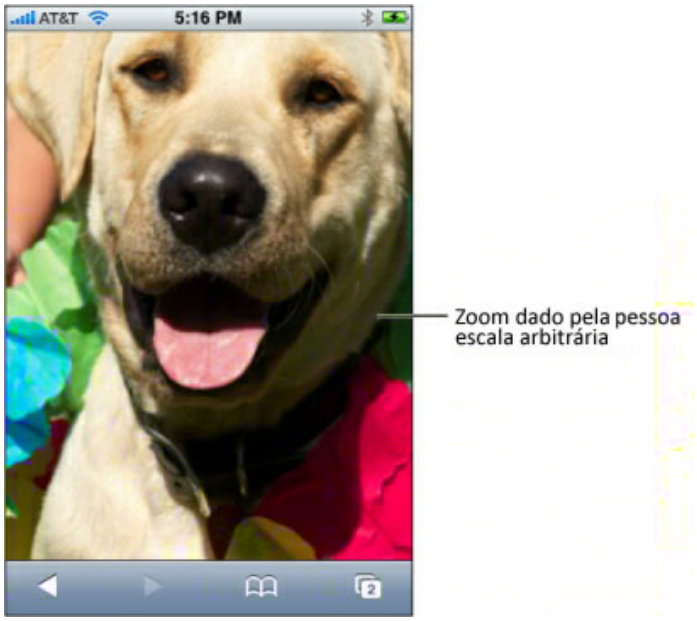



E, como citado, também é possível definir um *zoom* menor para a *viewport*.

```
<meta name="viewport" content="width=320,initial-scale=0.5">
```




Nos dispositivos móveis, a pessoa também pode fazer *zoom in* e *zoom out* usando gestos. Quando isso acontece, ela não altera o tamanho da *viewport*; altera sua **escala**! Consequentemente, os gestos de *pan* e *zoom* não alteram o layout da página.




## A conguração ideal de viewport

Observando a tabela anterior, é possível observar que diferentes configurações de *viewport* (e de *zoom* na *viewport*) produzem diferentes resultados e possibilidades de apresentação do site em condições controladas. Mas há uma questão que não ajuda em nada quando pensamos em possibilidades de uso de `width`: para contemplar todos os dispositivos móveis, seria preciso conhecer a largura de cada um deles!

O iPhone, nos exemplos de tela apresentados, tem largura de `320px`. Mas tomando como exemplo somente os *devices* exemplificados no gráfico de resolução de aparelhos, como visto na figura 2.1, seria muito trabalhoso identificar, marcar e fazer manutenções no código com todos os possíveis tamanhos.

Mas, não se preocupe, pois, mais uma vez, a tecnologia nos auxilia: existe uma forma de especificar automaticamente o tamanho do *device*. É como se estivéssemos programando e usando uma variável que, por padrão, já contém a largura do dispositivo que está realizando o acesso naquele momento. Basta usar:

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

Isso indica ao navegador que o `width` da *meta tag viewport* é o tamanho da largura do dispositivo!

Em condições normais, também é interessante que, assim que a página renderize, não haja alterações na escala inicial. Por isso, juntamente com `width=device-width` da *meta tag viewport*, é bom usar a `initial-scale` em `1`. E esta, para a maioria dos sites desenvolvidos segundo a filosofia do design responsivo, é a **configuração ideal de viewport**.

```
<meta name="viewport" content="width=device-width,initial-scale=1">
```

### Valores possíveis da meta tag

A seguir, veja a tabela - originalmente exibida no Nokia Developer (http://ow.ly/baLZP) - com os parâmetros possíveis da *meta tag viewport*, exemplos de valores, respectivos valores-padrão e a descrição de cada um deles.

| Parâmetro | Exemplos | Valores possíveis | Padrão | Descrição |
|---|---|---|---|---|
| width | width=device-width<br>width=320 | 1 a 10000 pixels ou device-width | device-width | Largura da viewport (em pixels) |
| height | height=device-height<br>height=640 | 1 a 10000 pixels ou device-height | device-height | Altura da viewport (em pixels) |
| initial-scale | initial-scale=0.5<br>initial-scale=2.0 | 0.1 to 10 | 1.0 (sem zoom) | Zoom para o primeiro carregamento da página (valores altos causam "zoom in"; valores baixos, "zoom out") |
| user-scalable | user-scalable=no | yes ou no | yes | Se a página permite zoom pelo usuário |
| minimum-scale | minimum-scale=0.5<br>minimum-scale=1 | 0.1 a 10 | 0.25 (ajuste pelo valor de width) | Limite do "zoom out" do usuário |
| maximum-scale | maximum-scale=1.5<br>maximum-scale=2.5 | 0.1 a 10 | 5 (ajuste pelo valor de width) | Limite do "zoom in" do usuário |
| target-densityDpi | target-densityDpi=high-dpi | device-dpi (sem ampliação), medium-dpi (zoom de 1.3) ou high-dpi (zoom de 1.5) | device-dpi | Modifica a densidade da tela, ampliando a página para uma maior resolução (DPI) |

Lembrando para tomar cuidado e prestar bastante atenção no uso combinado dos parâmetros possíveis para não causar efeitos desagradáveis e/ou não planejados. Por exemplo, ao especificar:

```
<meta name="viewport"
    content="width=device-width,initial-scale=1,maximum-scale=1">
```

Isso impossibilitaria a pessoa que está fazendo o acesso de dar *zoom* nas páginas do site. Se esse for realmente o objetivo proposto no site, tudo bem; mas, repetindo: atenção ao usar parâmetros combinados.

## 3.5 Convertendo um layout fixo em layout fluído

Construir layouts fluídos (ou grids flexíveis) é nosso maior objetivo ao ler este livro. Depois que os conceitos aqui mostrados já estiverem *internalizados* e bastante prática tiver sido feita, você conseguirá, antes mesmo de colocar o dedo para digitar a primeira regra CSS, já imaginar com foco à construção de layouts fluídos.

Enquanto isso não acontece, vamos fazer um exercícios interessante, que vai clarear bastante alguns dos principais conceitos de designs responsivos: vamos converter o layout fixo que vimos anteriormente para um layout fluído. Como? Simplesmente aplicando nossa fórmula mágica do web design responsivo! Mas primeiro...

Como vamos mexer mais com as medidas que especificamos anteriormente, não há necessidade de revisarmos todo o CSS mostrado. Ao invés disso, vamos nos ater aos trechos que mais interessam ao nosso exercício de conversão, começando com nosso `.container`.

A largura de `.container` é de `960px`. Obviamente, isso não ajuda em nada a montagem de um grid flexível dado que, como já foi visto, a principal e primeira medida a ser tomada para se construir um layout flexível é **usar somente medidas relativas no CSS**. Revisando nossa fórmula: **Alvo / Contexto = Resultado**.

No caso, como se trata do elemento contêiner, não temos um contexto que sirva como base. Então, da mesma forma que, arbitrariamente, especificamos essa largura para o elemento, vamos tentar uma largura com medida relativa que, quando vista no *browser*, se pareça o máximo possível com o valor que temos atualmente. Depois de alguns testes, a encontramos:

```
.container {
margin: 0 auto;
width: 67.5%; /* +/- 960 */
}
```

Como pôde ser visto neste sucinto trecho de código, uma excelente prática, quando estamos trabalhando com layouts fluídos, é fazer um comentário na frente do valor convertido (ou construído) para não perder a noção do que se está fazendo. Fica a dica.

Em relação a este valor de 67.5% para o contêiner, foi totalmente opcional e servindo somente para ilustrar nosso site-exemplo. Didaticamente, o Autor acredita ser interessante preservar a característica de *sites* centralizados para a adaptação e mais fácil assimilação dos conceitos de *web design* responsivo. Entretanto, outras abordagens poderiam ter sido usadas:

- Deixar o `width` em `960px` fixos e transformar todo o resto em `%` e `em`. Não tão eficiente, já que vai ficar com esta largura independentemente da tela ser maior ou menor que `960px`. Não seria uma solução responsiva.
- "Responsivar" pra telas *mobile* menores que `960px`, mas manter os `960px` para telas maiores. Usaria-se `max-width:960px` e continuaria igual para resoluções "*desktop-like*", mas, ao ser exibido em resoluções menores, vai se ajustar por não ter largura fixa.
- A solução anterior é responsiva para telas pequenas, mas limita em 960 pra telas maiores. Também é possível responsivar para todas, colocando `width:100%` no contêiner. Para projetos que precisam/queiram contar com largura total em quaisquer resoluções, esta abordagem seria a ideal.

Dando prosseguimento, agora é preciso passar o tamanho da fonte de `h1` para medidas relativas. A coisa está começando a ficar mais fácil já que, neste caso, temos todas variáveis da nossa fórmula! Lembre-se que existe um certo padrão entre os *browsers* de colocar o tamanho inicial das fontes em `16px`? Isso é tudo que precisamos no momento.

Quer dizer, `32` (**Alvo**) dividido por `16` (**Contexto**) é igual a `2` (**Resultado**) - lembrando daquela outra observação importante: quando se trata de medidas relativas para fontes, usamos a medida **em**; quando é para medidas de *layout*, **porcento**.

```
/* Antes */
h1 {
font-size: 32px;
}

/* Depois */
h1 {
font-size: 2em; /* 32 / 16 */
}
```

Depois disso, chegamos ao próximo elemento de nosso layout: `.content`. Usando nossa fórmula, novamente, ou seja: **15** dividido por **960** (a medida de nossa já finada largura fixa) é igual a **0.015625**. Mas, espere! Ao se aplicar este valor ínfimo ao layout, as coisas não funcionam como o esperado! Bem, como estamos lidando com porcentagens, precisamos **multiplicar o valor por 100**. *Matemágica* pura!

```
/* Antes */
.content {
margin: 15px 0;
}

/* Depois */
.content {
margin: 1.5625% 0; /* 15 / 960 */
}
```

Finalmente, nossas margens fluídas estão devidamente configuradas e prontas para o que der e vier! Com isso em mente, o elemento `.content-main` não vai representar nenhum perigo. Mas se você estiver, realmente, prestando atenção aos códigos de exemplo, poderá estar com uma pergunta intrigante em sua mente: onde está o **Contexto**?!

> **Como calcular o Contexto em elementos sem Contexto imediato**
>
> Quando o Contexto esperado não existe, isto é, o elemento-contêiner não apresenta largura definida, é preciso procurar no elemento imediatamente ascendente até que se encontre a definição de uma largura.

Ou seja, como o Contexto de `.content-main`, que é o elemento `.content`, não possui declaração explícita de largura, é preciso verificar o próximo elemento-ascendente. No caso, é o `.container`. Este sim com declaração (prévia) de largura de `960px`. Por isso, neste caso, o Contexto será de **960**:

```
/* Antes */
.content-main {
float: left;
width: 593px; /* Medida maior da Proporção Áurea aplicada ;-) */
```

```
}

/* Depois */
.content-main {
float: left;
width: 61.7708%; /* 593 (.content-main) / 960 (.container) */
}
```

O elemento `.hero` (e seu descendente `.brief`), dentro de `.content-main`, também precisa ter suas medidas convertidas. Algo interessante a se levar em conta é que o cálculo da relatividade de `.hero .brief` é `5 / 593 = 0.008431`. Contudo, apesar de esta ser uma ciência (pretensiosamente) exata, há muito do **fator humano** envolvido. Portanto, quando estiver desenvolvendo *grids* flexíveis e algum resultado, apesar de preciso, não for do agrado, não há nada de mais em (caso isso seja responsabilidade sua) arredondar este valor.

```
/* Antes */
.hero {
    margin: 25px 0;
}

.brief {
    margin: 5px 0;
}

/* Depois */
.hero {
    margin: 4.2158% 0; /* 25 / 593 */
}

.brief {
    margin: 1% 0; /* 5 / 593 */
    /* = 0.8431%, que, opcionalmente, pode ser arredondado para 1% */
}
```

Prosseguindo com a conversão dos outros elementos descendentes de `.content-main`:

```
/* Antes */
.last-contents {
font-size: 12px;
```

```
}

    .last-content-call {
    float: left;
    margin: 15px 15px 15px 0;
    width: 280px;
    }

        .last-content-call .brief {
        margin: 5px;
        }

/* Depois */
.last-contents {
font-size: .75em; /* 12 / 16 */
}

    .last-content-call {
    float: left;
    margin: 2.5295% 2.5295% 2.5295% 0; /* 15 / 593 (.content-main) */
    width: 47.2175%; /* 280 / 593 */
    }

        .last-content-call .brief {
        margin: 1.7857% 0; /* 5 / 280 */
        }
```

A conversão da barra lateral, com todos seus descendentes, fica assim:

```
/* Antes */
.content-sidebar {
background-color: #F0F0F0;
float: right;
padding: 10px;
width: 322px;
}

    .main-nav ul {
    list-style-type: none;
    }

        .main-nav li {
```

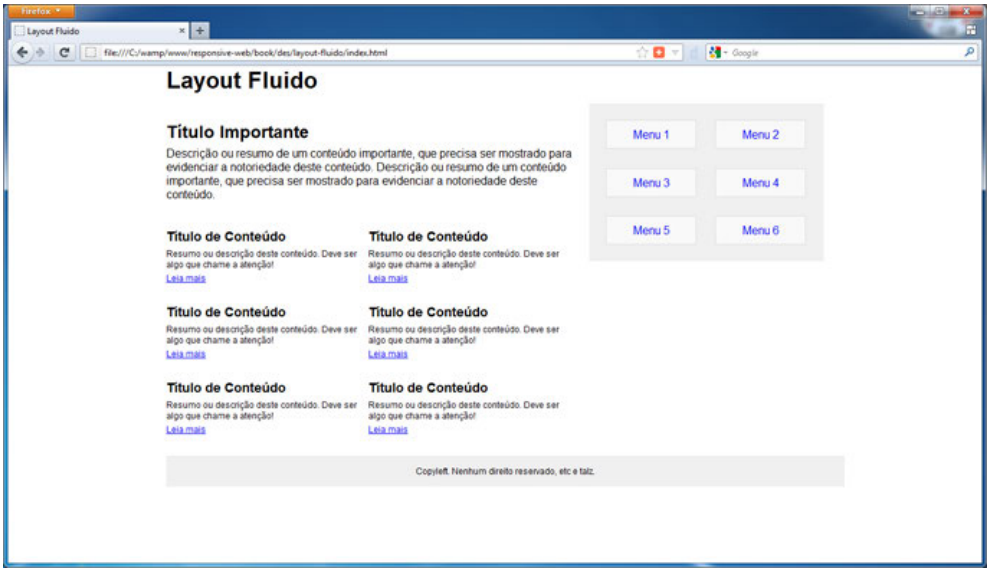
```
        background-color: #F9F9F9;
        float: left;
        margin: 15px;
        outline: 1px solid #DEDEDE;
        text-align: center;
        width: 130px;
        }

            .main-nav a {
            display: block;
            padding: 10px;
            text-decoration: none;
            }

/* Depois */
.content-sidebar {
background-color: #F0F0F0;
float: right;
padding: 1.0416%; /* 10 / 960 */
width: 33.5416%; /* 322 / 960 */
}

    .main-nav ul {
    list-style-type: none;
    }

        .main-nav li {
        background-color: #F9F9F9;
        float: left;
        margin: 4.6583%; /* 15 / 322 (.content-sidebar) */
        outline: 1px solid #DEDEDE;
        text-align: center;
        width: 40.3726%; /* 130 / 322 */
        }

            .main-nav a {
            display: block;
            padding: 7.6923%; /* 10 / 130 */
            text-decoration: none;
            }
```

Por fim, nosso humilde rodapé:

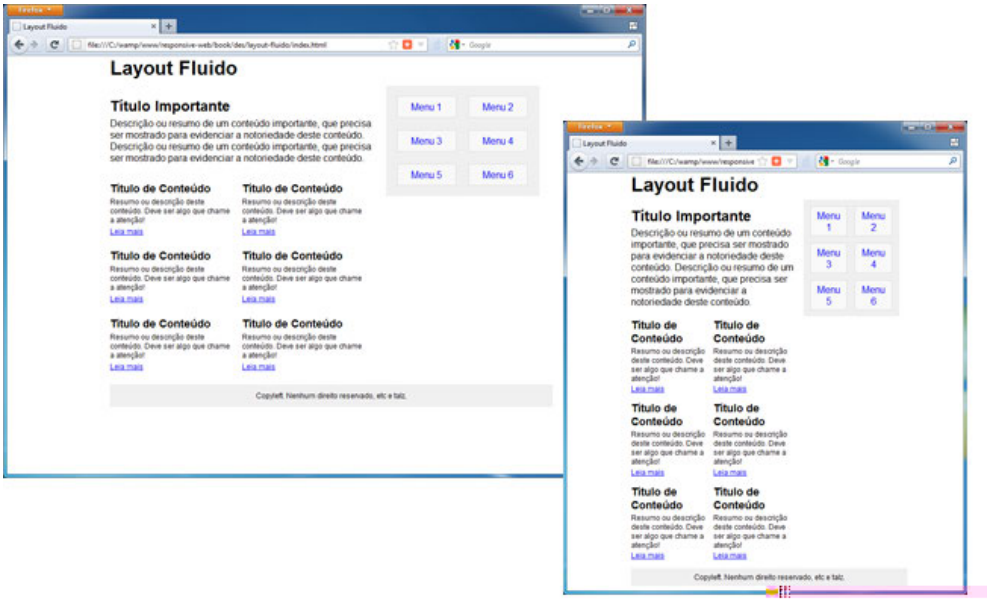
```
/* Antes */
.main-footer {
background-color: #F0F0F0;
clear: both;
float:left;
font-size: 12px;
margin: 15px 0;
padding: 15px;
text-align: center;
width: 100%;
}
```

```
/* Antes */
.main-footer {
background-color: #F0F0F0;
float:left;
font-size: .75em; /* 12 / 16 */
margin: 1.5625% 0; /* 15 / 960 */
padding: 1.5625%; /* 15 / 960 */
text-align: center;
width: 100%;
}
```

Ao pegar nosso CSS convertido e o abrir como estilização do HTML que já temos, num primeiro momento, não dá pra notar nada de mais. Com exceção de alguns poucos pixels pra lá ou pra cá, está a mesma coisa que visualizamos em nosso layout fixo.




Mas, ao redimensionar a janela, os caminhos da responsividade para a web começam a ser revelados:




Perceba que, ao redimensionar a janela do *browser* - e, claro, caso aconteça o

fato de alguma pessoa já acessar nossa página-exemplo usando resoluções das mais diversas -, o que acontece é que a apresentação está agradavelmente padronizada e **responsiva**!

Com isso você dá o primeiro grande passo para a construção de layouts responsivos para a web, meus parabéns!

Mas apesar de já ser um enorme passo em nossa jornada responsiva, nem somente de um grid flexível se dá o desenvolvimento de um site responsivo. Como vimos, ainda existe mais princípios de um web design responsivo a serem explorados: recursos flexíveis e Media Queries.

Capítulo 4

# Imagens e recursos flexíveis

## 4.1 CSS para imagens flexíveis

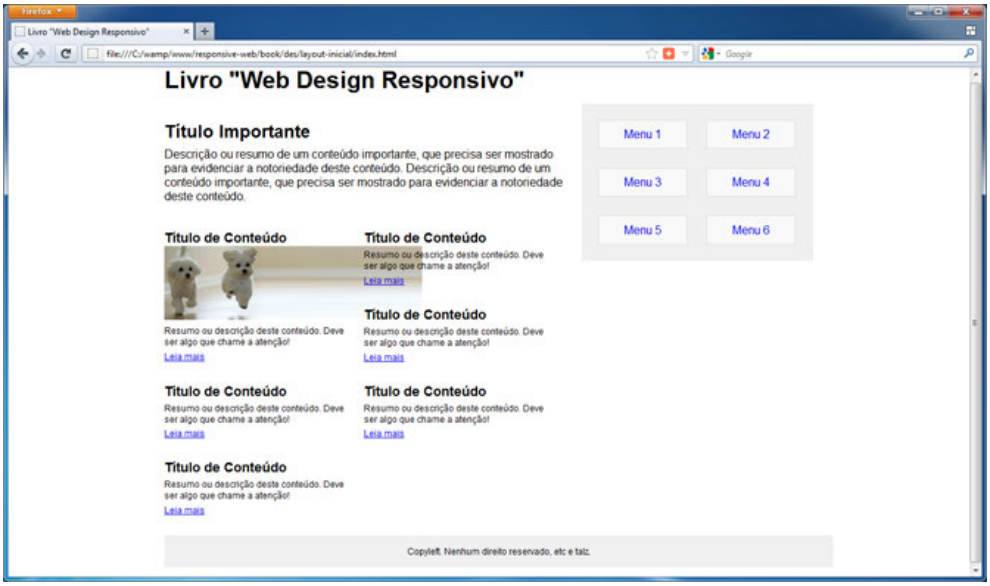
Continuando com o desenvolvimento de nosso HTML de exemplo, suponhamos que agora foi decidido que cada `.last-content-call` terá, logo abaixo do título, uma imagem para ilustrar seu conteúdo. Como já estamos num viés de desenvolvimento através da filosofia de um web design responsivo, é preciso encontrar uma solução que faça com que as imagens se adequem ao espaço devido e, independentemente da resolução, sejam bem apresentadas.

Para começar, vamos ver como ficará o exemplo com uma imagem colocada na primeira chamada. O HTML muda para:

```
<article class="last-content-call">
    <h2 class="secondary-title">
        Título de Conteúdo
    </h2>

    <img src="cute-dogs.jpg" width="350">
```

```
    <p class="brief">
        Resumo ou
        descrição deste conteúdo.
        Deve ser algo que chame a atenção!
    </p>

    <a href="#">Leia mais</a>
</article>
```

E o resultado é:



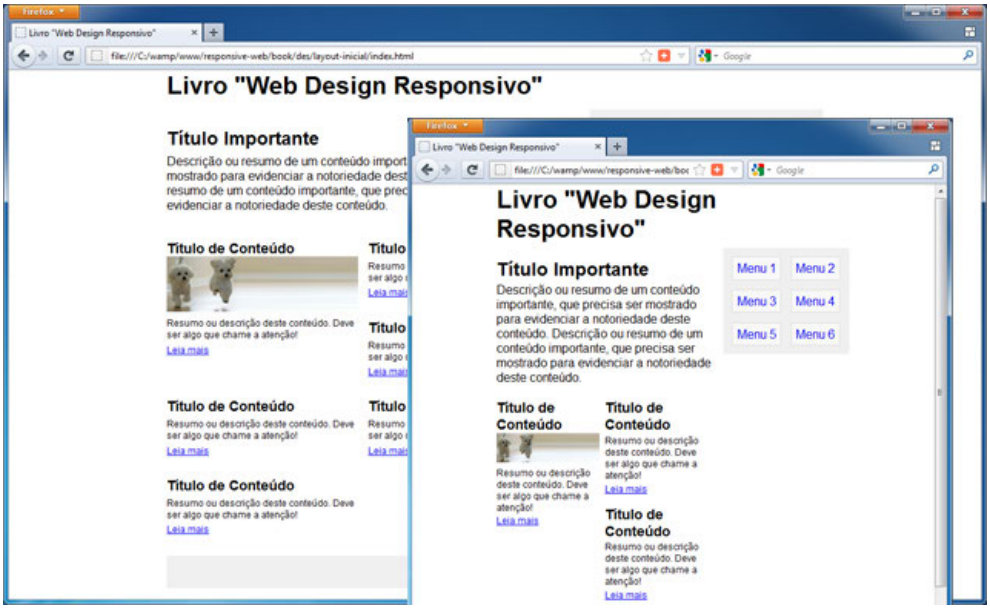

Veja que coisa: a imagem, corretamente marcada com seu `width` no HTML, possui 350px de largura. Acontece que, seja qual for a resolução em que a página estiver sendo vista, essa quebra acontecerá...


Pode parecer incrível mas, para conseguirmos um efeito interessante de imagens flexíveis, só é preciso um pouquinho de CSS. Nada de regras extravagantes, nem de propriedades recém-lançadas anteontem que nenhum *browser* ainda suporta; é o bom e velho CSS, com propriedades familiares a qualquer um com pouco tempo de estudos em folhas de estilo. Vamos acrescentar no início de nosso CSS:

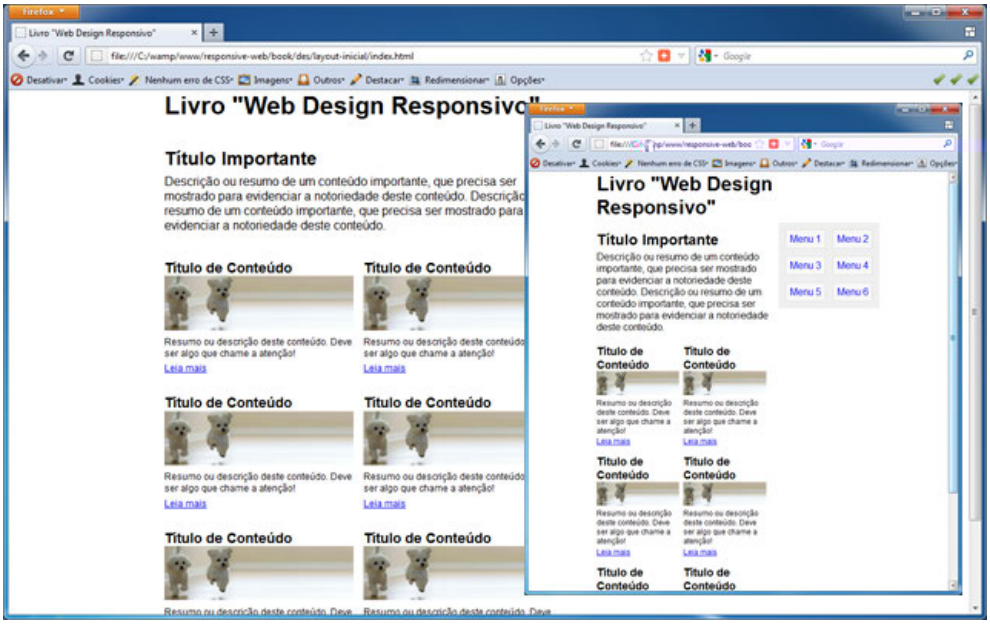
```
img {
    max-width: 100%;
}
```

Segundo a documentação sobre `max-width` do W3C (http://ow.ly/bcKdv), essa propriedade CSS permite restringir larguras de conteúdo dentro de um determinado intervalo. No caso, nosso "intervalo" foi 100%. Em outras palavras, nossa regra genérica de CSS significa algo como: "*as imagens podem ter qualquer largura até no máximo 100%*". Mas "100%" de quê? Do elemento em que estão contidas!

No caso da imagem dos cachorrinhos que usamos, ela é filha de `.last-content-call`. Portanto, a regra vale para a largura deste elemento ou, no caso de sua falta, na largura do elemento ascendente e assim por diante. Depois de aplicada a regra `max-width: 100%;` as coisas já começam a ficar mais interessantes:




Com essa regra em uso, já é possível complementar nosso layout fluído e ver como este fica com todos os `.last-content-call` com suas respectivas imagens:



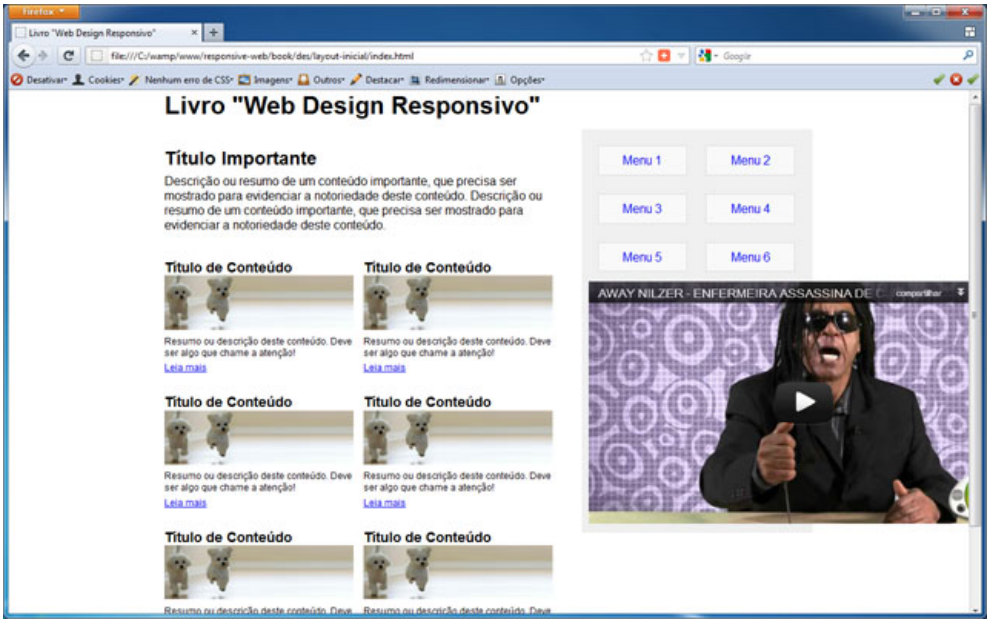
Uma observação importante é que, caso seja preciso se preocupar com o IE8 na implementação de imagens responsivas, é preciso colocar `width:auto` em imagens dentro de elementos com `float` sem declaração explícita de largura (http://ow.ly/cO3fP).

## 4.2 CSS para outros recursos flexíveis

E, mais uma vez, uma requisição foi feita à equipe de desenvolvimento do site. Dessa vez, será preciso que um vídeo seja sempre carregado na barra lateral, logo abaixo dos menu principal. Nosso `aside` então muda para o seguinte:

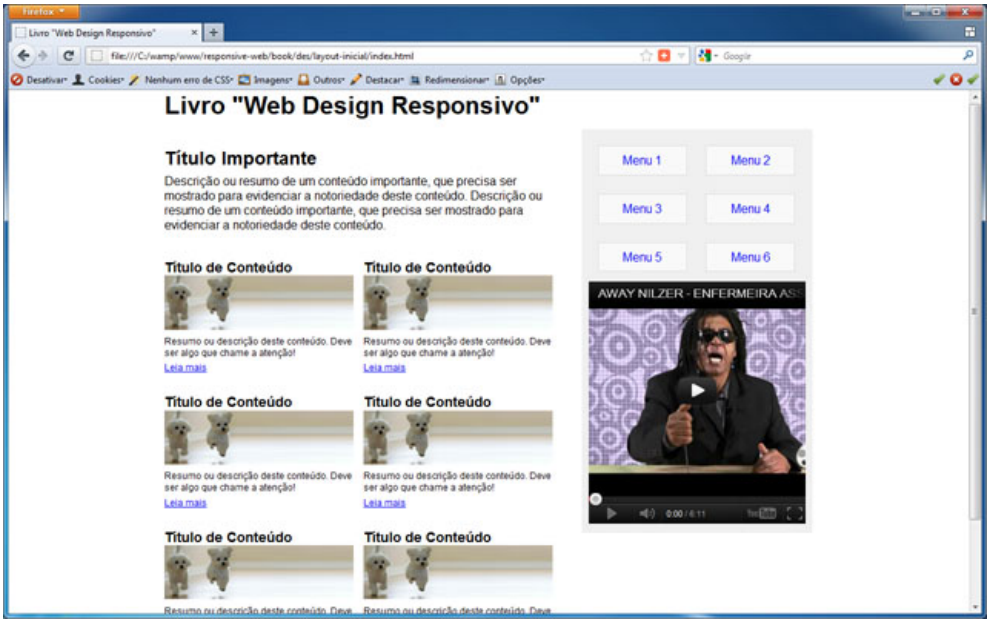
```
<aside class="content-sidebar">
    <nav class="main-nav">
        <ul>
            <li><a href="#">Menu 1</a></li>
            <li><a href="#">Menu 2</a></li>
            <li><a href="#">Menu 3</a></li>
            <li><a href="#">Menu 4</a></li>
            <li><a href="#">Menu 5</a></li>
            <li><a href="#">Menu 6</a></li>
        </ul>
    </nav>

    <iframe src="http://www.youtube.com/embed/1YLKyVwExOs" width="560"
    height="315" frameborder="0" allowfullscreen></iframe>
</aside>
```

O vídeo escolhido absolutamente não importa. O fato é que o código foi copiado diretamente do recurso de compartilhamento do YouTube (mas poderia ter sido qualquer outro serviço de compartilhamento de vídeos). Do jeito que de lá veio, cá está. Ao observamos o resultado disso, é possível perceber que algo não saiu como deveria.




O *embedded* está com o tamanho totalmente fora do aceitável para o caso e está quebrando o layout. No caso, ele não pode extrapolar os limites do `aside` e deve aparecer corretamente em qualquer resolução e dispositivo em que o site for acessado. Pode parecer incrível, mas a mesma regra que aplicamos anteriormente para imagens também pode ser aplicada para outros tipos de mídia e recursos, incluindo `iframe`, `object`, `embed` e `video`! Então basta atualizar para:

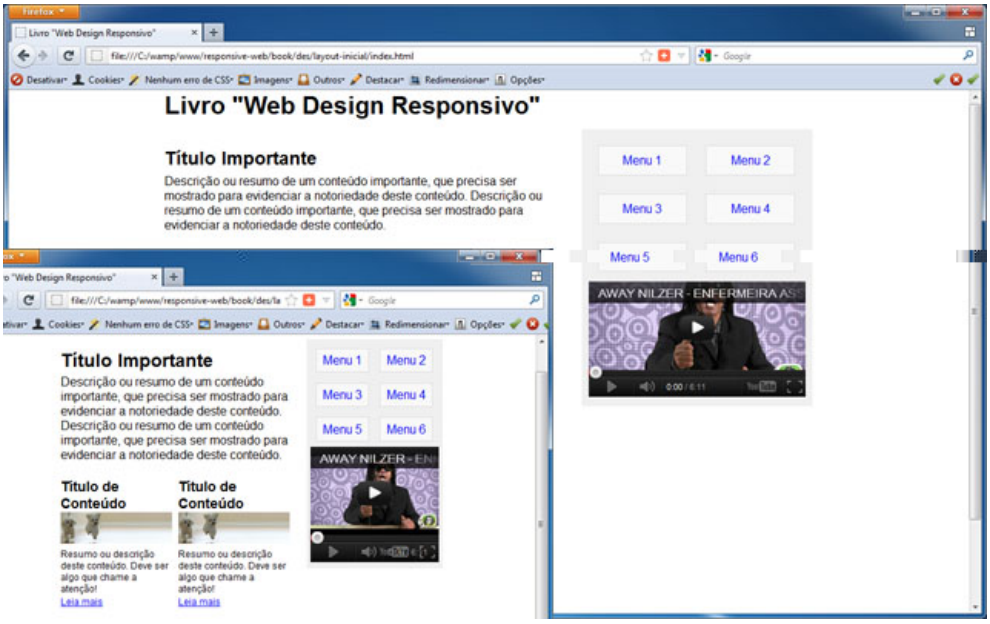
```
img,
iframe,
object,
embed,
video {
    max-width: 100%;
}
```




E, assim como o esperado, nosso HTML contempla boa apresentação/posicionamento em qualquer resolução. Em outras palavras, agora temos um **vídeo □exível**!

É possível ainda incrementar nossa regra. Nos testes realizados, a responsividade dos recursos não é comprometida de nenhuma maneira; o acréscimo facultativo só altera um pouco como os elementos são apresentados.

```
img,
iframe,
object,
embed,
video {
    height: auto;
    max-width: 100%;
}
```




Sem muitas alterações no comportamento dos elementos, é possível mudar o valor da propriedade `height` de `auto` para `100%` . Fica ao critério e/ou necessidades do projeto a decisão da mudança.

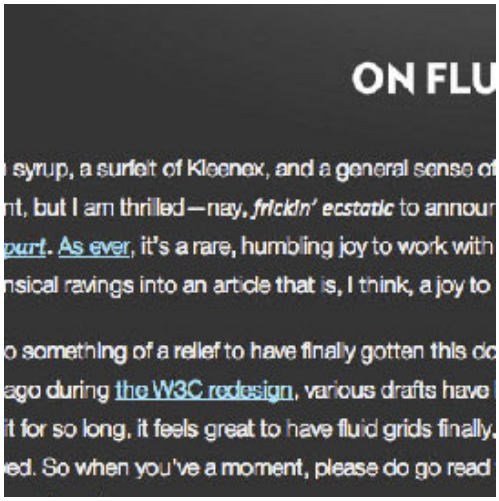
O YouTube coloca, automaticamente, tarjas pretas em cima e em baixo do vídeo ao se redimensionar, preservando o *ratio* original do vídeo. Entretanto, caso seja preciso usar vídeos de outras fontes (ou, até, mesmo, vídeos com hospedagem própria), existem algumas soluções:

- **Creating Intrinsic Ratios for Video.** Artigo muito interessante de **ierry Koblentz** que apresenta uma técnica que permite aos navegadores determinarem as dimensões de vídeos com base na largura do elemento que estão contidos: http://ow.ly/eqra4 .
- **FitVids.js.** Plugin jQuery simplíssimo de usar que dá a qualidade da responsividade a vídeos incorporados em qualquer página: http://ow.ly/eqrpN .
- **Fluid Width Video.** Artigo de **Chris Coyier** que aborda a inserção de vídeos com as *tags* `iframe`, `object` e `embed`: http://ow.ly/eqrT9 .

### Exceção à velharia: imagens redimensionadas com qualidade em IE6`

Aproveitem a exceção do que acabou de ser devidamente enfatizado: uma solução de como melhorar a qualidade de imagens redimensionadas exibidas em ambiente Windows (antes do 7), sobretudo com uso de IE6`.

Acontece que, usando a *web* sob estas condições, imagens redimensionadas ficam com a qualidade muito comprometida. Com a devida referência a Ethan Marcotte, reproduzo a imagem de exemplo em que o guru do web design responsivo mostra um exemplo desta questão.




Existem muitas boas tentativas por parte da comunidade para solucionar isso. Inclusive o próprio Ethan desenvolveu um JavaScript para tratar estas situações. Basicamente o script varre todo o documento, troca as imagens pelo famoso pixel transparente de 1px quadrado e aplica o filtro proprietário da Microsoft `AlphaImageLoader` em cada uma. A partir disso, sempre que a janela do navegador for redimensionada, o script recalcula, adequada e proporcionalmente, largura e altura das imagens, redimensionando o pixel *spacer*. A seguir, o script de Ethan.

```
var imgSizer = {
    Config : {
        imgCache : []
        ,spacer : "/path/to/your/spacer.gif"
    }
```

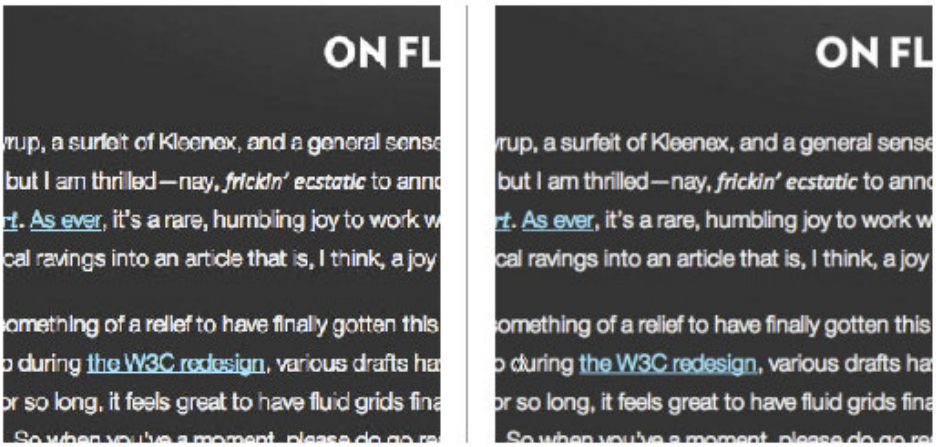
```
,collate : function(aScope) {
    var isOldIE =
        (document.all && !window.opera && !window.XDomainRequest)
        ? 1 : 0;

    if (isOldIE && document.getElementsByTagName) {
        var c = imgSizer;
        var imgCache = c.Config.imgCache;

        var images = (aScope && aScope.length) ? aScope :
        document.getElementsByTagName("img");
        for (var i = 0; i < images.length; i++) {
            images[i].origWidth = images[i].offsetWidth;
            images[i].origHeight = images[i].offsetHeight;

            imgCache.push(images[i]);
            c.ieAlpha(images[i]);
            images[i].style.width = "100%";
        }

        if (imgCache.length) {
            c.resize(function() {
                for (var i = 0; i < imgCache.length; i++) {
                    var ratio = (imgCache[i].offsetWidth /
                    imgCache[i].origWidth);
                    imgCache[i].style.height =
                        (imgCache[i].origHeight * ratio) + "px";
                }
            });
        }
    }
}

,ieAlpha : function(img) {
    var c = imgSizer;
    if (img.oldSrc) {
        img.src = img.oldSrc;
    }
    var src = img.src;
    img.style.width = img.offsetWidth + "px";
    img.style.height = img.offsetHeight + "px";
```

```
        img.style.filter =
            "progid:DXImageTransform.Microsoft.AlphaImageLoader
        (src='" + src + "', sizingMethod='scale')"
        img.oldSrc = src;
        img.src = c.Config.spacer;
    }

    // Ghettomodified version of Simon Willison's addLoadEvent() --
    http://simonwillison.net/2004/May/26/addLoadEvent/
    ,resize : function(func) {
        var oldonresize = window.onresize;
        if (typeof window.onresize != 'function') {
            window.onresize = func;
        } else {
            window.onresize = function() {
                if (oldonresize) {
                    oldonresize();
                }
                func();
            }
        }
    }
}
```

E, através de um comentário condicional, fazer o script entrar em ação somente quando o browser for Internet Explorer:

```
<!--[if IE]>
    <script src="imgsizer.js"></script>
<![endif]-->
```

Depois da implementação do script, a diferença na qualidade das imagens renderizadas sob as condições expostas é nítida:




Frequentemente, a partir de sugestões e casos da comunidade, o script é revisado e atualizado. Portanto, passe a acompanhar o artigo em que, originalmente, Ethan disponibilizou o script, "Fluid Images" (http://ow.ly/bg5hO), e aproveite para conhecer mais detalhes e pormenores de como a técnica foi desenvolvida e como funciona.

## 4.3 O problema de imagens em layouts fluídos

Com as técnicas apresentadas para imagens flexíveis em web designs responsivos, é possível fazer um bom número de testes para fixar os conhecimentos, pois isso já é a **base** a respeito de gráficos para a responsividade na web. Entretanto, surge uma questão (se não, **a** questão) sobre apresentar imagens em qualquer resolução: **peso** e **tamanho**!

Quando o acesso é tradicional, feito via *desktop* ou *notebook*, é justificável supor que o usuário está vendo a página através de um monitor de tamanho relativamente confortável. Isso, juntamente com o fato de que a velocidade média de conexão à internet no Brasil é de 512Kb a 2Mb (http://ow.ly/bg78j), não traz maiores problemas para a pessoa que navega em sites com imagens.

Já quando o acesso segue a tendência crescente de acesso *mobile*, temos questões e problemas seríssimos. Esses aparelhos costumam ter uma tela menor e, muitas vezes, resoluções não muito boas. Além disso, a velocidade da conexão móvel é inferior a de outros meios de acesso à Internet. Com tudo isso, temos que pensar a respeito

de como servir imagens adequadamente aos visitantes, levando em conta seu **peso** (tamanho, em KB, das imagens) e seu **tamanho** (número de pixels que formam a imagem).

Segundo as técnicas mostradas até então, uma mesma imagem é apresentada seja pra qual resolução for e, conforme o caso, esta é redimensionada para se adaptar à realidade da resolução do *device* que está sendo usado. Isso que revela um ponto crítico: o **mesmo arquivo de imagem** que é apresentado numa boa velocidade de conexão, vista em uma tela de tamanho considerável, também é o servido e renderizado em dispositivos móveis!

Mesmo que sem querer, pessoas utilizando dispositivos *mobile* que acessam o site serão penalizadas por não disporem de um *display* maior e/ou uma conexão mais veloz no momento do acesso. O cidadão acessando seu site pseudo-otimizado para dispositivos móveis será obrigado a esperar carregar aquela imagem de 1500px, pesando 80Kb, só para vê-la sendo lentamente renderizada, já redimensionada (e distorcida), demorando muito mais tempo para terminar de ser baixada e exibida em sua conexão 3G do que ele para decidir procurar o que precisa no site da concorrente...

Se o acesso é feito via *mobile*, é mais lógico (e melhor para quem acessa) que uma versão reduzida da imagem, de menor peso e tamanho, seja enviada a fim de aprimorar a experiência da visita. Afinal, **71%** das pessoas esperam que o acesso a um site móvel seja tão rápido quanto no desktop (http://ow.ly/bg7Wp)!

## 4.4 Técnicas para imagens flexíveis em web designs responsivos

Tendo visto o básico necessário para apresentar imagens flexíveis em designs responsivos para a web, cabe ressaltar que existem diversas técnicas para que essa entrega de imagens seja a mais adequada possível (dentro da perspectiva de quem desenvolveu a respectiva técnica, evidentemente). A seguir, é possível conhecer uma série de técnicas para imagens flexíveis; como sempre, não existe "a melhor" técnica, e sim a que mais se enquadra ao seu estilo de desenvolvimento, dentro de um projeto específico.

### Riloadr

Riloadr se auto-intitula um framework independente carregador de imagens responsivas. Ele realmente possui muitas boas características. Só para citar algumas:

- Relativamente fácil de aprender e usar;
- Oferece bastante controle sobre como as imagens responsivas irão se comportar;
- Opção de carregamento sob demanda ("lazy load") das imagens;
- Suporta browsers com JavaScript desabilitado (ou sem JavaScript);
- Não faz múltiplas requisições para a mesma imagem.

A ideia principal do Riloadr é deixar o atributo `src` completamente fora das *tags* de imagem, substituindo-o pelo atributo `data-src`. Através da mágica do JavaScript, é possível obter resultados satisfatórios com marcação do tipo:

```
<img data-src="pessoa.jpg">
<noscript>
    <img src="img/small/pessoa.jpg">
</noscript>
```

Realmente, o Riloadr apresenta uma das técnicas mais eficientes para imagens responsivas ou, pelo menos, é uma das que mais oferecem opções de personalização e customizações para este objetivo. Para ver a documentação completa e exemplos de uso, visite o repositório oficial do Riloadr (http://ow.ly/cO3HZ).

### jQuery Picture

jQuery Picture é um plugin para jQuery (de somente 2KB) para adicionar suporte a imagens responsivas nos *layouts*. Ele suporta o elemento `<figure>` (usando alguns `data-*`) e, até, no elemento proposto `<picture>` (http://ow.ly/cO5lY). Um exemplo de uso seria:

```
<figure class="responsive" data-media="assets/img/small.png"
    data-media440="assets/img/medium.png"
    data-media600="assets/img/large.png">
    <noscript>
        <img src="assets/img/large.png" alt="">
    </noscript>
</figure>
```

E, seguindo a melhor forma enxuta de iniciar plugins com jQuery:

```
$(function(){
    $('figure.responsive').picture();
})
```

A boa notícia é que, além de ser bastante simples e fácil de se usar, ele tem suporte pelos melhores navegadores e pelo Internet Explorer a partir de sua versão 9. Para maiores informações, dê uma olhada no respositório oficial do jQuery Picture (http://ow.ly/cO4OW).

### Picture□ll

Picturefill é uma abordagem de imagens responsivas que é possível utilizar atualmente sem maiores complicações (em *browsers* com suporte a Media Queries, que será abordado futuramente). Para tanto, ele exige uma marcação diferente da usual para imagens:

]

```
<div data-picture data-alt="Descrição da imagem">
    <div data-src="small.jpg"></div>

    <div data-src="medium.jpg" data-media="(min-width: 400px)">
    </div>

    <div data-src="large.jpg" data-media="(min-width: 800px)">
    </div>

    <div data-src="extralarge.jpg" data-media="(min-width: 1000px)">
    </div>

    <!-- Conteúdo para JavaScript desabilitado/não suportado -->
    <noscript><img src="assets/img/small.jpg" alt="Descrição da imagem">
    </noscript>
</div>
```

Apesar da marcação incomum (e do trabalho de manutenção que esta acarreta), a proposta é bem interessante e, para mais detalhes a respeito, basta acessar o repositório do Picturefill (http://ow.ly/cO5En).

### Adaptive Images

**Adaptive Images**, de Matt Wilcox (http://ow.ly/cNXkD), é uma das técnicas que, na opinião do criador (e de milhares de pessoas pelo mundo que a utilizam), mais fazem mais sentido. Adaptive Images detecta o tamanho da tela do visitante e automaticamente cria, faz cache e entrega versões redimensionadas de imagens referenciadas em tags `<img>`, apropriadas ao dispositivo que está sendo usado.

A técnica envolve o uso de PHP, JavaScript e um pouco de conteúdo em seu arquivo `.htaccess`. Além de ser bastante eficiente por servir diferentes tamanhos da mesma imagem dependendo do dispositivo usado para a visualização do site, um outro ponto forte é que nenhuma alteração na marcação HTML se faz necessária.

Para instruções mais detalhadas e como instalar e usar a técnica, visite o site oficial da Adaptive Images (http://ow.ly/cNVks).

### Qual técnica para imagens responsivas usar?

Estas foram somente algumas recomendações de técnicas para imagens responsivas possíveis. Existem, literalmente dezenas delas e, a cada dia que passa, mais pessoas desenvolvem novas técnicas, fazem propostas de padrões de marcação e tentam desenvolver soluções eficientes para a questão de imagens responsivas para web.

E, dentre tantas técnicas diferentes, fica a pergunta: qual dela usar? Como já comentado, não existe uma resposta única para esta questão, e sim uma análise do projeto em que se está envolvido para que a decisão seja tomada. Para ajudar nesta decisão, alguma considerações devem ser feitas:

- É importante que funcione sem JavaScript?
- Sendo JavaScript, pode ser um plugin ou deve ser uma solução “nativa"?
- Pode-se usar marcação especial?
- Semântica é essencial?
- As páginas precisam ser validadas no validador do W3C?
- Tem-se acesso a arquivos server side e .htaccess?
- Deve-se primar pela performance?

Respondendo a essas perguntas e tendo em mente que as pessoas esperam que o acesso a um site móvel seja tão rápido quanto no desktop (http://ow.ly/bg7Wp), é possível escolher uma solução que seja a melhor para o projeto, em específico, que se esteja trabalhando no momento. Existe uma planilha pública com uma compilação de algumas soluções possíveis para a questão das imagens responsivas. É interessante que você dê uma olhada e, quem sabe, já encontre alguma solução que se adeque a seu projeto atual: http://ow.ly/cO758.

### Nas palavras de Ethan Marcotte...

Para fechar com chave de ouro esta questão importantíssima sobre imagens responsivas, cabe lermos (e pesquisarmos e refletirmos) sobre as palavras do próprio Ethan Marcotte quando participou de uma entrevista à .NET Magazine (http://ow.ly/cSCTR), quando o entrevistador, Oliver Lindberg, disse "*Eu quero saber qual a abordagem mais prática e robusta para exibição de imagens responsivas em vários dispositivos*". A resposta foi:

*Bem, há muita atividade em toda a área de "imagens responsivas" agora. Detecção de largura de banda é um ponto cego para todos nós se estamos trabalhando no front-end ou backend.* `navigator.connection` *não está amplamente implementado ou (eu diria) é especí co o su ciente para ser útil e todos nós estamos assistindo a Device APIs Working Group (http://ow.ly/cSDta) com interesse.*

*Dito isto, eu estou realmente animado com a discussão [do elemento] "picture", que tem sido habilmente conduzida por Mat Marquis (http://ow.ly/cSEky) no W3C community group for responsive images (http://ow.ly/cSEc0). E a solução 'Picture ll', de Scott Jehl, signi ca que podemos começar a experimentar isso hoje.*

*Claro que ainda não temos, sequer, uma especi cação para um novo padrão de marcação, muito menos uma solução nativa funcional. Então, talvez a curto prazo, precisemos con ar em negociação de conteúdo server-side - embora, se esse for o caso, espero que haja uma mentalidade mais "mobile rst", à Brett Jankord's Categorizr (http://ow.ly/cSFcA).*

*Mas eu pergunto se há outra opção disponível para nós: por que não podemos perguntar aos nossos usuários o que eles preferem? Eles estão acostumados a tomar decisões "cosméticas" de UI sempre que visitam o Gmail ou a escolher a qualidade de vídeo no YouTube. Cada vez que clicam em um link "móvel" ou "desktop" em seus telefones, eles estão optando por experiências e conjuntos de recursos que são radicalmente diferentes um do outro. Então, eu estou querendo saber se há uma solução mais "matizada" aqui: poderíamos pedir aos nossos usuários para, bem, dizer-nos que tipo de*

*decisões de largura de banda tomar?*

### O futuro

A maioria das soluções apresentadas anteriormente possuem algum ponto falho; afinal, assim como o próprio *web design* responsivo, as técnicas que lhe são inerentes ainda estão no início, não estão "maduras" e ainda não se chegou a uma solução "perfeita". A boa notícia é que já existem propostas no W3C para tentar preencher esta lacuna, com o atributo `srcset` no HTML (http://ow.ly/eqwpe) e a propriedade `image-set` para CSS (http://ow.ly/eqwjs).

Esperemos somente mais algum tempo e usar imagens em *web design* responsivo será tão simples quanto escrever algumas linhas de HTML ou CSS atualmente!

## 4.5 Imagens em alta resolução

Devido à tecnologia atual, somente alguns dispositivos possuem *displays* de alta resolução, capazes de apresentar corretamente imagens de altíssima resolução. No capítulo com referências de recursos para web design responsivo 7, você vai encontrar soluções para servir imagens de alta resolução para dispositivos que possuem *retina display*, o que é bem interessante, mas é preciso ter em mente que a tendência é que mais aparelhos comecem a apresentar uma tela com essas características.

Portanto, é importante ter conhecimentos de alguns pontos cruciais para aqueles projetos em que imagens de alta resolução estão contempladas pelo planejamento. **Adam Bradley** apresenta diretrizes de muito interesse sobre a questão (http://ow.ly/eotkE) que, independentemente de qual solução se opte por usar, devem ser levadas em consideração:

**Não fazer requisição da mesma imagens várias vezes.** Quando o navegador vê pela primeira vez um elemento `img` no DOM, ele, imediatamente faz uma requisição ao servidor para esta imagem (mesmo antes de o DOM estar pronto). O atributo `src` deve ser alterado antes mesmo de o navegador passar pelo elemento de imagem ou já será tarde e a requisição já terá sido feita. Além disso, se vários pedidos estavam sendo feitos para uma imagem, isso não só retarda o navegador e impacta na performance, mas, também, aumenta o "estresse" adicional sobre os servidores.

**Não usar cookie nem o elemento <base>.** Não se deve usar por causa do *look-ahead parser* dos navegadores que já disparam as requisições das `<img>` antes mesmo de o JavaScript entrar em ação. Ou seja, o *download* da imagem começa antes de o código que especifica o `cookie` ou `base` rodar.

**Manipulação mínima do DOM.** Manipular o DOM requer tempo e retarda a atuação do navegador, portanto, a solução usada deve fazer tão poucas atualizações no DOM quanto possível para garantir que todos os dispositivos, especialmente os de baixa capacidade de processamento, possam lidar com esta manipulação.

**Imagens devem continuar visíveis sem JavaScript.** Se o JavaScript não estiver habilitado, o navegador ainda deve ser capaz de interpretar e renderizar todas as imagens da página em seu tamanho *default*.

**Não apresentar imagens em alta resolução para todos.** A maioria dos usuários não têm *displays* de alta resolução, isso é fato. Ter servidores servindo imagens maiores para todos os usuários adiciona uma "pressão extra" nos servidores, clientes, largura de banda, etc. Só apresente imagens de alta resolução para aqueles que realmente podem se valer deste recurso.

Como dito, estas são somente diretrizes, não regras rígidas e imutáveis de procedimentos. Então, sempre que for possível, levando em consideração a parte técnica e de planejamento, seguir estas dicas trará benefícios aos projetos que se valem de imagens em alta resolução.

## 4.6 Tipos de imagem para web

Sim, foram (e continuam sendo) desenvolvidas técnicas que visam sanar o problema de apresentar uma imagem que melhor condiz ao *device* que a pessoa usa ao visitar um site. Mas um passo anterior a isso é saber escolher qual **tipo de imagem** usar em qual situação.

Atualmente, 3 são os principais formatos de imagem usados na web - GIF, JPG e PNG - e conhecer as diferentes características desses formatos de imagem mais comuns para poder optar conscientemente entre um ou outro, conforme a necessidade, é o primeiro passo rumo à otimização de um site - não somente os de web design responsivo!

### GIF

**GIF** quer dizer *Graphics Interchange Format* (Formato de intercâmbio de gráficos). Imagens no formato são conhecidas como "*indexed*", dada a característica do formato de suportar até 256 cores (8 bits) *indexadas* numa paleta, provendo imagens de pequeno peso, bastante usadas e ideais para o ambiente *online* e usado, principalmente, para ícones, ilustrações, logos, imagens "chapadas" e qualquer outra necessidade que possa ser suprida com as características inerentes ao formato.

GIF é um formato sem perda, o que significa que, quando você modifica e salva a imagem, não há mudança na qualidade. GIF suporta "transparência binária", ou seja, pixels em uma imagem GIF ou são totalmente transparentes ou totalmente opacos. Além disso, o formato, depois de uma revisão poucos anos depois de seu lançamento, passou a suportar animações, o que consolidou seu sucesso à época da, então novidade e, certamente, contribuiu, juntamente com suas outras boas características, para a "consagração" do formato e amplo uso na web.

Devido a suas características, o uso do formato GIF já foi indicado para:

- Logos
- Ícones
- Desenhos
- Ilustrações simples
- Imagens com poucas cores

Atualmente, devido ao advento do PNG 8 (que será visto a seguir), o uso do formato GIF deve se dar mais para imagens animadas.

### JPG (ou JPEG)

**JPEG** é o acrônimo de *Joint Photographic Experts Group* e é um dos mais usados no mundo não somente em ambientes virtuais, mas também por câmeras digitais e dispositivos diversos de captura de imagem. E esse índice de uso não é em vão, já

que o formato, de 24 bits, permite o uso de até **16 milhões de cores**, o que garante imagens de boa excelente qualidade.

Apesar de o JPG poder ter milhões de cores e ainda conseguir uma excelente taxa de compressão, diferentemente do GIF, é um formato **com perda**. Ou seja, cada vez que se salva o arquivo, a compressão é refeita e um pouco de qualidade se perde. A vantagem é que é possível escolher o nível de qualidade das imagens salvas em JPG, conforme a necessidade. A mesma imagem com uma qualidade excepcional que pode servir para impressão de um banner, após ser salva com qualidade inferior, também pode servir para exibição e transferência na web.

Infelizmente, o formato não aceita transparência, mas, devido às características mostradas, seu uso mais indicado é:

- Fotografias
- Imagens com muitas cores
- Imagens de alta complexidade
- Aliar qualidade e peso do arquivo
- Uso da mesma imagem em vários meios de acesso (ajustando a qualidade)

## PNG

**PNG**, ou *Portable Network Graphics*, é o mais recente formato de imagem dos mais comuns usados na web e, com o objetivo de suprir determinadas restrições (técnicas e legais) do GIF, o formato trouxe consigo características desejáveis do GIF

e JPG. O formato pode vir como **PNG 8** bits ou **PNG 24** bits, cada um tendo suas características próprias e servindo para determinados propósitos.

PNG-8 pega muito do GIF, sendo um formato sem perda, provendo suporte a 256 cores e transparência binária (sendo que consegue gerar arquivos de peso menor que GIF).

PNG-24 junta "o melhor dos mundos", já que é um formato sem perda de qualidade, aceita 16 milhões de cores e, uma das mais interessantes, aceita diferentes níveis de transparência! Obviamente tudo isso acarreta em uma imagem de peso maior.

Dependendo se se vai usar PNG8 ou PNG24, suas recomendações de uso são, respectivamente, iguais à dos formatos GIF e JPG - lembrando que para imagens complexas ou de muitas cores que exigem transparência, PNG24 é o ideal.

### SVG

**SVG** é uma linguagem de marcação para descrever gráficos bidimensionais e imagens. Segundo a Wikipédia (http://ow.ly/ev4Ba):

*SVG é a abreviatura de Scalable Vector Graphics que pode ser traduzido do inglês como grá cos vetoriais escaláveis. Trata-se de uma linguagem XML para descrever de forma vetorial desenhos e grá cos bidimensionais, quer de forma estática, quer dinâmica ou animada. Umas das principais características dos grá cos vetoriais, é que não perdem qualidade ao serem ampliados.*

E preste uma atenção especial na parte "não perdem qualidade ao serem ampliados"! Exatamente o que é preciso quando pensamos em gráficos que precisam ser bem apresentados em diversas resoluções diretentes, não é verdade?

Por ser independente de resolução, SVG é ótimo para a apresentação de gráficos na web. SVG, sendo uma descrição XML para gráficos e imagens, é capaz de trabalhar bem com formas, caminhos, cores, gradientes e fontes. Filtros e animação também podem ser definidos. Uma vez criadas, imagens SVG podem ser usadas em qualquer lugar, em qualquer escala e resolução.

A grande vantagem de usar SVG em web designs responsivos é que, apesar de existirem diversas técnicas para imagens flexíveis , não é preciso se preocupar em usar versões de tamanhos diferentes de uma mesma imagem já que, conforme explicado, trata-se de um formato vetorial, que pode ser redimensionado à vontade sem que sua qualidade seja comprometida.

Você vai conseguir apresentar bem SVG nativamente em diversos navegadores, como:

- Chrome 4+
- Opera 9.5+
- Firefox 4+
- Safari 4+
- Internet Explorer 9+

## Icon Fonts

**Icon Fonts** na verdade não é um formato de imagem; como sugere o próprio nome, são gráficos (geralmente, ícones) que, na verdade, são fontes! Trata-se de uma técnica inteligentíssima para prover "imagens" que são bem apresentadas em quaisquer resoluções sem a necessidade de ser preciso diferentes versões, em tamanhos diferentes, da mesma imagem ou gráfico.

Por exemplo, quando é preciso usar vários ícones num *site*, geralmente um arquivo é feito e usado como *sprite*. Usando Icon Fonts, você incorpora em suas páginas, através da propriedade @font-face de CSS (http://ow.ly/ev7lV), fontes "especiais", em que cada caractere é, na verdade, um ícone! E como, apesar de se parecer com uma imagem, trata-se de uma fonte, além do conveniente de se poder aumentar e diminuir à vontade seu tamanho sem perda de qualidade, também é possível aplicar todas as propriedades CSS relativas à fontes, o que dá uma flexibilidade incrível na hora da apresentação e efeitos!

Para se ter uma noção, dê uma olhada no Icon Fonts are Awesome (http://ow.ly/ev7Sj), que é um demo de Icon Font quer permite alterar, em tempo real, algumas propriedades CSS para testes, tais como tamanho, cor e posisionamento e formato de sombra.

Projetos de renome, como **Twitter Bootstrap**, se valem desta técnica para conferir mais dinamismo e modernidade em seus recursos. Provavelmente você deveria fazer o mesmo! Aqui vão algumas referências para você começar:

- **Glyphicons.** Excelentes *set*s de Icon Fonts gratuitos: http://ow.ly/ev9us .
- **Font Squirrel @font-face Generator.** Tenha certeza de que suas fontes serão incorporadas e visualizadas corretamente em qualquer navegador: http://ow.ly/ev9mX .
- **Free Icon Fonts for Web User Interfaces.** Coletânea incrível de vários *site*s que disponibilizam Icon Fonts: http://ow.ly/ev9Jb .

### Breves considerações sobre formatos de imagem

Num primeiro momento, pode parecer um tanto que inútil abordar os formatos de imagens mais comuns usados na web num livro sobre web design responsivo. Entretanto, pensando melhor, é muito importante saber escolher, conscientemente, qual é o melhor formato para as diferentes necessidades dos projetos.

Usando imagens adequadas e otimizadas para cada situação (por exemplo, JPG para fotografias, PNG para ícones, etc), é possível tirar melhor proveito das características técnicas das imagens e, ao não usar formatos inadequados, muitos recursos - em diversos sentidos da palavra - são poupados. Você deveria, também, passar as imagens do site em otimizadores de imagens (que fazem o peso da imagem fique menor sem alterar sua qualidade), já que, com menos KB para ser baixado no momento do acesso, o tempo total de carregamento é menor. Existem

muitas boas ferramentas para isso, como jpegtran (http://ow.ly/dxSyx), JPEGmini (http://ow.ly/dxSte), TinyPNG (http://ow.ly/dxSpB), OptiPNG (http://ow.ly/dxSmr) e Smush.it (http://ow.ly/dxSjc).

Outra dica importantíssima é o uso de *sprites* (http://ow.ly/dxRdy), uma técnica para combinação de imagens menores em uma imagem maior. Para mostrar a imagem correta, é preciso ajustar os valores da propriedade CSS `background-position`. Combinando várias imagens desta forma, é possível reduzir os pedidos HTTP (e reduzir requisições HTTP é um importantíssimo passo rumo à melhora de performance e desempenho de um site).


É possível fazer *sprites* "manualmente", mas há ferramentas online para isso, como CSS Sprite Generator (http://ow.ly/dxRt7), que dá a opção de fazer upload de imagens para serem combinados em um único sprite, gerando o código CSS (os valores de background-position) para renderizar as imagens.

Prover imagens otimizadas para cada contexto/necessidade é um passo importante para a otimização, em geral, de *web sites*. Existem muitas outras técnicas para conseguir os melhores resultados de cada formato apresentado; técnicas estas que extrapolam o objetivo deste livro, mas, certamente, devem ser de interesse para qualquer um que procure, sempre, prover melhores experiências aos visitantes de seus sites, garantindo que seus projetos sejam rápidos e super otimizados.

Capítulo 5

# Media Queries

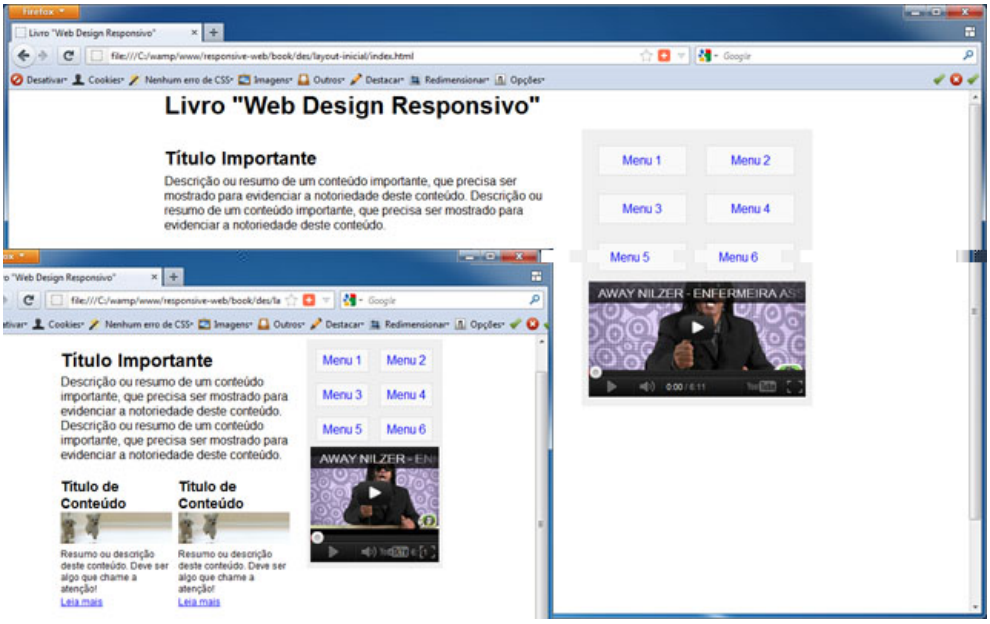
No capítulo anterior, foi apresentada a técnica essencial para se obter imagens (e outros recursos) responsivas na web.

Como foi citado no capítulo 2, quando foi abordada a trinca tecnológica do web design responsivo (seção02.2), 3 são as tecnologias necessárias para tal. Já foram abordados o layout fluído e as imagens (e outros recursos) flexíveis. E, finalmente, temos as **Media Queries**, uma peça chave no nosso quebra-cabeça.

Vamos conhecê-las e, depois, aplicá-las na prática em nosso site de exemplo.

## 5.1 Primeiro, os Media Types

Até onde chegamos com nosso site exemplo, já conseguimos fazer o layout ser flexível, ajustando a largura dos elementos conforme o tamanho da tela.




Mas, quando o assunto é desenvolver sites de qualidade que se adaptem a todo e qualquer dispositivo, é preciso ir um passo além. É preciso que o site não somente tenha seus elementos flexíveis, mas também que esses elementos possam variar de posição, tamanho, se escondam ou apareçam, conforme a necessidade/display. **É preciso que o design se ajuste**!

E é exatamente neste ponto que entram as *Media Queries*. Mas, antes de adentrarmos no fascinante estudo das *Media Queries*, é interessante entender um pouco sobre **Media types**.

*Media Types* é uma recomendação da W3C (http://ow.ly/cQ3MF) desde o CSS2. Com Media Types, é possível apresentar o site de maneira diferente, dependendo da *media*. É possível uma apresentação diferenciada (através de folhas de estilo) quando a página está sendo vista de um projetor, que pode ser diferentes de quando se usa uma impressora, um sintetizador de voz, uma TV, dentre outros. Resumidamente, eis a lista:

- **all**. A folha de estilo serve para todos os dipositivos.
- **braille**. Para dar *feedback* quando se usa algum dispositivo tátil.
- **embossed**. Impressoras em braille paginadas.

- **handheld**. Dispositivos móveis (comumente com tela pequena e largura de banda limitada).
- **print**. Para material paginado e para documentos visualizados na tela no modo Visualização de Impressão.
- **projection**. Destinado a apresentações projetadas como, por exemplo, projetores.
- **screen**. Destinado, principalmente, para telas coloridas de computador.
- **speech**. Para sintetizadores de voz.
- **tty**. Dipositivos de "grade fixa" para exibição de caracteres, com o teletypes, terminais e alguns outros.
- **tv**. Aparelho "tipo TV" (baixa resolução, cores, *scroll* limitado, som).

A ideia é que, ao ser acessada, a página *web* seja renderizada conforme o estilo mais apropriado ao *device* que está sendo usado para realizar o acesso, ou seja, se a pessoa está usando um *handheld*, o estilo especificado para este dispositivo, em específico, entra em ação; se é através de uma TV, o estilo apropriado à TVs é usado; e assim por diante.

Vamos a um exemplo de implementação de Media Types na `<head>` de uma página (a especificação da `media` deve ser em minúsculo):

```
<link rel="stylesheet" type="text/css" media="print"
                                       href="print_style.css">
```

Quer dizer, caso a *tag* `link` tenha o atributo de *Media Type* especificando *print*, somente quando alguém quiser imprimir o(s) documento(s) que levam esta *tag* é que `print_style.css` entrará em ação; para os demais tipos de *media*, não.

Outro exemplo seria caso se quisesse usar a mesma folha de estilo para diferentes *medias*:

```
<link rel="stylesheet" type="text/css" media="print, handheld"
href="print_handheld_style.css">
```

Neste caso, a(s) página(s) somente seria renderizada se valendo de `print_handheld_style.css` caso o dispositivo fosse um *handheld* ou no caso de a pessoa querer imprimí-la(s).

Também é possível, dentro de um mesmo arquivo de estilos, especificar regras para diferentes *medias* assim (e, obviamente, as regras que não ficam dentro de algum bloco `@media` são aplicadas a todos os dispositivos):

```
@media print {
    body {
    font-size: 10pt;
    }
}

@media screen {
    body {
    font-size: 13px;
    }
}

@media screen, print {
    body {
    line-height: 1.2;
    }
}
```

Mas, com a evolução das tecnologias de desenvolvimento *web*, as diferenças entre os tipos de *devices*, antes tão discrepantes, começaram a desaparecer. Como escreveu **Diego Eis** em seu artigo sobre Media Queries (http://ow.ly/cQ85G):

*Cada vez mais surgem dispositivos de diversos tamanhos com hardwares bem parecidos com os desktops. Isso faz com que a navegação destes aparelhos tenha uma experiência muito próxima de desktop. Um exemplo atual é o iPhone. Sua tela tem boa qualidade e seu navegador renderiza as páginas como um navegador normal de desktop. Logo, não tem motivo para prepararmos um layout e um CSS com media type HANDHELD para o iPhone. Apesar de ele ser um handheld, ele não trabalha como um. Contudo, ele também não trabalha como um desktop. Mesmo a renderização do MobileSafari sendo idêntica a de um desktop, o comportamento do usuário e a forma de navegação são diferentes. Logo, temos um meio termo. Não podemos disponibilizar um CSS para HANDHELD, nem um CSS totalmente SCREEN.*

A solução? **Media Queries**.

## 5.2 Media Queries

Quer dizer, começou a não ficar tão clara a diferença entre os vários tipos de *devices* e suas capacidades, deixando os *Media Types* numa posição de, sozinhos, não atenderem mais às necessidades de desenvolvimento, já que muitos, como o citado iPhone, acabaram ficando numa "zona cinzenta" de *medias*, tendo características tanto de *handheld*, quanto de *desktop*.

A solução? **Media Queries**. Como muito bem argumentou **Sérgio Lopes** em seu artigo sobre *Media Queries* (http://ow.ly/cQ8Di), *Melhor do que separar os dispositivos em desktop (screen) e mobile (handheld), os novos* Media Queries *permitem que foquemos principalmente no tamanho da tela onde vamos exibir o conteúdo*.

Ou seja, *Media Queries* - que já é uma recomendação do W3C desde junho de 2012 (http://ow.ly/cUjIw) - tornaram possível a evolução para um novo nível de especificação de estilos para *web sites*. Por exemplo, é possível indicar um CSS somente para *devices* com tela de até `320px`:

```
<link rel="stylesheet" media="screen and (max-width: 320px)"
                       href="320.css">
```

Caso se queira usar várias declarações em somente um arquivo, podemos fazer:

```
@media screen and (min-width: 320px) {
    body {
    font-size: 80%;
    }
}

@media screen and (min-width: 480px) {
    body {
    font-size: 90%;
    }
}
```

Mas não somente isso. Existe uma gama extensa de parâmetros de *Media Queries*.

## 5.3 Parâmetros para trabalhar com Media Queries

Seguindo alguma das sintaxes apresentadas anteriormente, é possível especificar o uso de estilos separados por arquivo ou blocos dentro do mesmo arquivo usando um

ou vários dos parâmetros a seguir. Importante ressaltar que a maioria dos parâmetros pode ser prefixado com `min-` ou `max-`, que representam, respectivamente, "maior ou igual a" e "menor que ou igual a" (assim como foi mostrado no exemplo anterior).

Vamos ver alguns desses parâmetros:

### aspect-ratio

- **Valor**: <proporção>
- **Media**: visual, tátil
- **Aceita min/max**: sim

Descreve a proporção da área de exibição do *browser* usado. O valor é composto de dois números inteiros positivos, separados por um caractere de barra ("/"). Isso representa a proporção de pixels na horizontal (primeiro termo) para pixels na vertical (segundo termo).

Por exemplo, para aplicar regras com a janela do navegador em *widescreen* de proporção 16:9:

```
@media screen and (aspect-ratio: 16/9) {
    [...]
}
```

### color

- **Valor**: <cor>
- **Media**: visual
- **Aceita min/max**: sim

Indica o número de bits por componente de cor do dispositivo. Se ele não é um dispositivo de cor, o valor é igual a 0. Se os componentes de cor têm diferentes números de bits, o menor número é usado. Por exemplo, se um *display* utiliza 5 bits para azul e vermelho e 6 bits para verde, o dispositivo vai usar 5 bits. Se o dispositivo usa cores indexadas, o número mínimo de bits por componente de cor na tabela de cores é usado.

Por exemplo, para especificar regras para dispositivos coloridos:

```
@media all and (color) {
    [...]
}
```

Ou, para estilos que somente serão aplicados em dispositivos com, pelo menos, 4 bits de cor:

```
@media all and (min-color: 4) {
    [...]
}
```

Ou ainda com no máximo 16 bits de cores:

```
@media (max-color: 16) {
    [...]
}
```

### color-index

- **Valor**: <inteiro>
- **Media**: visual
- **Aceita min/max**: sim

Descreve o número de entradas na tabela de cores do dispositivo. Se o dispositivo não possuir tabela de cores, então o valor é 0.

Por exemplo, para especificar que algumas regras serão aplicadas a todos *devices* com cores indexadas:

```
@media all and (color-index) {
    [...]
}
```

Dispositivos com, pelo menos, 2 cores indexadas:

```
@media (min-color-index: 2) {
    [...]
}
```

Ou incorporar uma folha de estilos para, somente, dispositivos com menos de 256 cores indexadas:

```
<link rel="stylesheet" media="all and (max-color-index: 256)"
href="http://foo.bar.com/stylesheet.css" />
```

### device-aspect-ratio

- **Valor**: <proporção>
- **Media**: visual, tátil
- **Aceita min/max**: sim

Descreve a proporção do *display*

### device-width

- **Valor**: <comprimento>
- **Media**: visual, tátil
- **Aceita min/max**: sim

Descreve a largura, em pixels, do *display* do aparelho. Para meios contínuos, é a largura da tela; para meios paginados, é a largura do tamanho da folha de cada página. Obviamente, <comprimento> não pode ser um valor negativo.

Por exemplo, aplicar uma folha de estilo a um documento quando visualizado numa tela menor que 800px de largura:

```
<link rel="stylesheet" media="screen and (max-device-width: 799px)"
href="style.css">
```

### grid

- **Valor**: <inteiro>
- **Media**: todas
- **Aceita min/max**: não

Determina se se trata de um dispositivo de grade ou um dispositivo de mapa de bits (bitmap). Se o dispositivo é baseado em grade (tal como um terminal TTY ou um visor do telefone, com apenas uma fonte), o valor é 1; caso contrário, é 0.

Por exemplo, para aplicar algumas regras a todos dispositivos assim:

```
@media (grid) {
    [...]
}
```

Para aplicar um estilo em dispositivos móveis com *display* de 15 caracteres ou menos:

```
@media handheld and (grid) and (max-width: 15em) {
    [...]
}
```

Note que a unidade "em" tem um significado especial para dispositivos assim, já que a largura exata de um "em" não pode ser determinada, assume-se que 1em é a largura de uma célula da grade horizontal e a altura de uma célula vertical.

### height

- **Valor**: <comprimento>
- **Media**: visual, tátil
- **Aceita min/max**: sim

Altura, em pixels, da janela do navegador sendo usado. Para meios contínuos, é a altura da *viewport*, incluindo o tamanho da barra de rolagem; para meios paginados, é a largura da "page box" da impressora. <comprimento> não pode ser um valor negativo.

Por exemplo, especificar algumas regras CSS para dispositivos com a altura da janela do *browser* em 300px:

```
@media screen and @media (height: 300px) {
    [...]
}
```

Ou com a altura menor que 600px:

```
@media screen and @media (max-height: 599px) {
    [...]
}
```

### monochrome

- **Valor**: <inteiro>
- **Media**: visual
- **Aceita min/max**: sim

Indica o número de bits por pixel em dispositivos monocromáticos (escala de cinza). Se o dispositivo não for monocromático, o valor é zero.

Por exemplo, para aplicar estilos em qualquer dispositivo monocromático:

```
@media all and (monochrome) {
    [...]
}
```

Para aplicar estilos em qualquer dispositivo monocromático com, pelo menos, 2 bits por pixel:

```
@media all and (min-monochrome: 2) {
    [...]
}
```

### orientation

- **Valor**: landscape | portrait
- **Media**: visual
- **Aceita min/max**: não

Indica se o dispositivo está em modo "paisagem" (tela é mais larga que alta) ou "retrato" (tela é mais alta que larga).

Então, para se aplicar regras para dispositivos que estão no modo "paisagem", basta usar:

```
@media all and (orientation:landscape) {
    [...]
}
```

### resolution

- **Valor**: <resolução>
- **Media**: bitmap
- **Aceita min/max**: sim

Trata sobre a resolução (densidade de pixels) do dispositivo, que pode ser especificada em pontos por polegada (dpi) ou pontos por centímetro (dpcm). Por exemplo:

```
@media (resolution: 72dpi) {
    [...]
}

@media (min-resolution: 118dpcm) {
    [...]
}
```

Uma <resolução> sem algum dos prefixos "min-" ou "max-" nunca vai atuar em *devices* com pixels não-quadrados (*non-square pixels*).

### scan

- **Valor**: progressive | interlace
- **Media**: tv
- **Aceita min/max**: no

Trata sobre o processo de escaneamento que um dispositivo do tipo "TV" pode fazer. Por exemplo, para aplicar uma folha de estilo apenas para televisões com escaneamento progressivo:

```
@media tv and (scan: progressive) {
    [...]
}
```

### width

- **Valor**: <comprimento>
- **Media**: visual, tátil
- **Aceita min/max**: sim

Largura, em pixels, da janela do navegador sendo usado. Para meios contínuos, é a largura da *viewport*, incluindo o tamanho da barra de rolagem; para meios paginados, é a largura da "page box" da impressora. <comprimento> não pode ser um valor negativo.

Por exemplo, especificar algumas regras CSS para dispositivos com a largura da janela do *browser* em 640px:

```
@media screen and @media (width: 640px) {
    [...]
}
```

Ou com a largura menor que 768px:

```
@media screen and @media (max-width: 767px) {
    [...]
}
```

## 5.4 Operadores Lógicos

Apesar de alguns dos Operadores Lógicos de Media Queries já terem sidos mostrados em exemplos anteriores, para ficar claro sobre sua existência, possibilidades e utilidade, conforme o W3C (http://ow.ly/cQVfk), os Operadores Lógicos de Media Queries são:

- "and" e "or"
- "not"
- "only"

### and e or

Várias Media Queries podem ser combinadas em uma lista (separada por vírgulas) de consultas. Se uma ou mais das consultas são verdadeiras, toda a lista é verdadeira; do contrário, é falsa. Na sintaxe de media queries, a palavra `and`" expressa o operador lógico **AND** e a `vírgula` expressa o operador lógico **OR**.

Por exemplo:

```
/* Telas e impressões coloridos */
@media screen and (color), print and (color) {
    ...
}
```

### not

O operador lógico **NOT** é indicado pela palavra `not`. A presença da palavra `not` no início da consulta nega o resultado, ou seja, se a consulta de mídia for verdadeira, sem o "not" ela se torna falsa (e vice-versa). Os navegadores que suportam somente

Media Types não irão reconhecer a palavra-chave `not` e a folha de estilo associada, portanto, não será aplicada. Exemplo:

```
<link rel="stylesheet" media="not screen and (color)"
                       href="style.css">
```

#### "only"

A palavra-chave `only` pode ser usada para esconder as folhas de estilo de agentes de usuário mais antigos. Os agentes devem começar a processar as consultas de mídia com `only` como se a palavra-chave `only` não estivesse presente.

```
<link rel="stylesheet" media="only screen and (color)"
                       href="example.css">
```

## 5.5 Vá quebrando seu design em breakpoints bem pensados

Apesar de existir mais de uma dezena de parâmetros para Media Queries, a verdade é que somente alguns poucos são usados. Isso ocorre não pela falta de cuidado ou competência no desenvolvimento, mas principalmente devido ao fato de que as necessidades atuais da maioria dos sites não requerem o uso de muitos dos parâmetros. De fato, a maioria dos utilizados atualmente dizem respeito à largura e altura do *display* e à orientação do dispositivo.

E, como foi visto anteriormente, é possível declarar regras CSS específicas conforme o tamanho do dispositivo ou *browser*, através dos parâmetros `device-width` e `width` (e seus prefixos). Isso leva a uma discussão importante sobre quais são os *breakpoints* (ou "pontos de interrupção") mais eficientes, quer dizer, os "pontos" em que se deve tratar o *web design* de forma diferenciada para que a experiência de quem está visitando o site seja a melhor e mais eficiente possível.

Qual é o momento de parar de escrever regras CSS para uma resolução e começar para outra? Se estou escrevendo estilos que atendam a diversos tipos de *devices*, em qual momento, especificamente, é o "limite da resolução" de um dispositivo e início de outro? Em que momento devemos pegar nosso sidebar e jogá-lo para baixo do nosso conteúdo principal? Quando a largura do dispositivo obrigado ao usuário scrollar, por exemplo! Isso definiria um **breakpoint** (ou "pontos de interrupção").

*Breakpoints* são os delimitadores das regras de CSS para atenderem a diferentes especificações (feitas por você, mesmo, ao usar os parâmetros de *Media Queries*),

quer dizer, os "pontos" em que se deve utilizar estilos diferentes para tratar o *web design* de forma diferenciada para que a experiência de quem está visitante o site seja a melhor/mais eficiente possível.

### Diferentes abordagens sobre breakpoints

Quando os estudos de web design responsivo estavam bem no início, alguns desenvolvedores propuseram o uso de *breakpoints* conforme a resolução que determinados dispositivos tinham. Por exemplo, `@media only screen and (min-device-width : 320px) and (max-device-width : 480px)` para *smartphones*, `@media only screen and (min-width : 1224px)` para notebooks e desktops e assim por diante. Como já vimos, mal isso começou e foi constatado que essa abordagem não funciona muito bem, já que, conforme a evolução dos dispositivos, é praticamente impossível determinar qual *device* se está usando conforme a resolução no momento da visita ao site.

Portanto, não existe consenso sobre quais seriam os melhores ou mais eficientes *breakpoints* em *web design* responsivo. Pra dizer a verdade, existe bastante discrepância de opiniões, alguns chegando a afirmar que não seria preciso a especificação de *breakpoints*, mas sim o desenvolvimento de *layouts* que se adaptariam a quaisquer resoluções.

Para os que pensam de uma maneira diferentes, a preferência é se valer do recurso dos *breakpoints* para alterar a disposição do *layout* conforme o dispositivo (ou largura do *browser*) que se esteja usando no momento. Para estes casos, existem posições diferentes sobre quais seriam os melhores (ou preferencialmente usados) *breakpoints*.

Realmente não é possível especificar um conjunto de *breakpoints* que sejam "os melhores". Cada projeto *web* tem suas próprias peculiaridades e pede uma implementação diferenciada dos demais. Alguns tentaram chegar a um "ponto de partida genérico", especificando *breakpoints-chave*, que podem facilitar o início de qualquer projeto. Vamos ver os que são mais conhecidos:

### 320 and Up

320 and Up (http://ow.ly/cSIXX):

```
@media print { }

@media only screen and (min-width: 480px) { ... }
```

```
@media only screen and (min-width: 600px) { ... }

@media only screen and (min-width: 768px) { ... }

@media only screen and (min-width: 992px) { ... }

@media only screen and (min-width: 1382px) { ... }

@media only screen and (-webkit-min-device-pixel-ratio: 1.5),
    only screen and (min--moz-device-pixel-ratio: 1.5),
    only screen and (min-device-pixel-ratio: 1.5)
{ ... }
```

Perceba que o uso de *Media Queries* se dá de forma "crescente", ou seja, primeiro contemplando estilos para dipositivos com resoluções menores com `480px` e `600px` (geralmente, *smartphones* e *netbooks*), para depois abordar os de maior resolução, como os tradicionais PCs e dispositivos com alta resolução (por exemplo, as *Smart TVs* mais atuais).

### Less Framework

Less Framework (http://ow.ly/cSQIb):

```
/* Tablet Layout */
@media only screen and (min-width: 768px)
                   and (max-width: 991px) { ... }

/* Mobile Layout */
@media only screen and (max-width: 767px) { ... }

/* Layout largo de mobile */
@media only screen and (min-width: 480px)
                   and (max-width: 767px) { ... }

/* Retina display */
@media
only screen and (-webkit-min-device-pixel-ratio: 2),
only screen and (min-device-pixel-ratio: 2) { ... }
```

Os próprios comentários do Less Framework já ajudam bastante a identificar para quais tipos de navegação as folhas de estilo foram projetadas. Os desenvolvedo-

res definiram os *breakpoints* no que chamaram de "Tablet Layout", "Mobile Layout", "Layout largo de mobile" e "Retina display" (regras estas, obviamente, somente para os poucos dispositivos que temos hoje que possuem esta característica).

### Skeleton

Skeleton (http://ow.ly/cSQ78):

```
/* Menor que 960 */
@media only screen and (max-width: 959px) { ... }

/* Tablet Portrait ao padrão 960 */
@media only screen and (min-width: 768px)
                   and (max-width: 959px) { ... }

/* Todos tamanhos de mobile */
@media only screen and (max-width: 767px) { ... }

/* Mobile em landscape a tablet Portrait */
@media only screen and (min-width: 480px)
                   and (max-width: 767px) { ... }

/* Mobile em portrait a mobile em landscape */
@media only screen and (max-width: 479px) { ... }
```

O *Skeleton* é bem específico ao propor *breakpoints* para dispositivos levando em consideração, inclusive, a orientação usada no momento (*landscape* ou *portrait*). Quer dizer, um planejamento da experiência do usuário mais detalhado e bem feito que, se bem usado, tem maiores chances de promover uma boa experiência aos visitantes e garantir uma boa dose de "agradabilidade".

### Twitter Bootstrap

Twitter Bootstrap (http://ow.ly/cSLyw):

```
/* Telefones em landscape e abaixo */
@media (max-width: 480px) { ... }

/* Telefone em landscape a tablet em portrait */
@media (max-width: 767px) { ... }
```

```
/* tablet em portrait a landscape e desktop */
@media (min-width: 768px) and (max-width: 979px) { ... }

 /* Desktop grande */
@media (min-width: 1200px) { ... }
```

O *Twitter Bootstrap* também teve o cuidado de projetar para vários dispositivos usando as variações de orientação. Apesar de conter menos *breakpoints* que o *Skeleton*, dá conta do recado tranquilamente (lembrando que isso é o que vem por default e, seja lá em qual deles você esteja se inspirando, pode acrescentar ou retirar quantos *breakpoints* quiser).

## 5.6 Gerenciamento de erros

Vimos quais os tipos possíveis de parâmetros para Media Queries e como é possível, através de operadores lógicos, "refinar" consultas a mídias específicas. Mas, e quando as coisas dão errado? E se, por acaso, ocorrer algum erro de sintaxe ou algum parâmetro inexistente for especificado?

Tipos de mídia desconhecidos serão avaliados como `false`. Efetivamente, eles são tratados de forma idêntica para os tipos conhecidos de mídia que não correspondem ao tipo de mídia do dispositivo. Por exemplo, a consulta de mídia `unknown` vai ser avaliada como `false`, a não ser que `unknown` seja um tipo de mídia suportada.

### Parâmetros de Media Queries desconhecidos

Os agentes de usuário representarão uma consulta de mídia como `not all` quando algum dos parâmetros de mídia não for conhecido. Por exemplo:

```
<link rel="stylesheet" media="screen and (max-weight: 3kg)
                                  and (color), (color)"
href="style.css">
```

A primeira consulta será representada como `not all` e avaliada como falsa; a segunda, como se a primeira não tivesse sido especificada.

Veja um outro exemplo que resultará em `not all` e tente compreender o motivo:

```
@media (min-orientation:portrait) {
    [...]
}
```

Adivinhou? O parâmetro `orientation` não admite prefixo `min-`!

### Valores desconhecidos em parâmetros de Media Queries

Tal como acontece com parâmetros desconhecidos, agentes de usuário interpretam como `not all` quando um dos **valores** de algum parâmetro especificado não for conhecido. Por exemplo, essas duas consultas seriam interpretadas como `not all`:

```
@media (color: 20example) {
    [...]
}

@media (min-width: -100px) {
    [...]
}
```

### Media Queries mal formadas (erros de sintaxe)

Os agentes de usuário conseguem lidar com símbolos inesperadas encontrados ao analisar uma consulta de mídia. Eles "lêem" até o final da consulta observando correspondência de pares de `()`, `[]`, `{}`, `""` e `''` e lidando corretamente com escapes. Consultas de mídia com *tokens* inesperados são interpretadas como `not all`.

```
/* Aplicado somente a dispositivos "speech" */
@media (example, all,), speech {
    [...]
}

/* Aplicado somente a dispositivos "screen" */
@media &test, screen {
    [...]
}

/* Sem espaço entre "and" e "(" */
@media all and(color) {
    [...]
}

/* Ponto e vírgula?! */
@media test;,all {
    [...]
}
```

## 5.7 Uso consciente de Media Queries

Muita coisa foi vista sobre Media Queries. Evidentemente, não se espera que, numa primeira lida de todo esse conteúdo, tudo seja entendido e absorvido. Na verdade, a melhor maneira para entender e fixar tudo isso é colocando alguns projetos em prática!

Mas, mesmo antes de sequer começar a mexer com as fantásticas Media Queries, é preciso levar em consideração algumas questões muito importantes sobre seu uso, tais como se usar arquivos externos ou não, a melhor maneira de usar *breakpoints*, se é melhor começar pelos tamanhos menores ou maiores, etc. Há algum conteúdo sobre isso circulando na *web* e, felizmente, é possível encontrar algumas considerações importantes no blog do zomig (http://ow.ly/cTV4J). Vamos ver algumas delas.

### Media Queries em arquivos separados ou num único arquivo

Segundo foi visto anteriormente, é possível realizar consultas de mídia chamando folhas de estilo em separado ou, dentro de um mesmo arquivo, realizar a diferenciação de estilos usando *breakpoints*.

```
<link rel="stylesheet" type="text/css"
                        media="screen and (min-width:480px)"
href="480.css">
<link rel="stylesheet" type="text/css"
                        media="screen and (min-width:768px)"
href="768.css">
```

```
@media only screen and (min-width: 480px) { ... }

@media only screen and (min-width: 768px) { ... }
```

Mas qual é a melhor maneira de usar? Chamar vários arquivos externos, um para cada *breakpoint*, ou somente um arquivo com todos *breakpoints* especificados? Na verdade, a resposta não é tão simples e há algumas considerações a esse respeito.

Quando se usa chamar vários arquivos diferentes, algo importante a ser levado em conta é que *browsers* que não oferecen suporte a Media Queries não irão baixar os arquivos. Também, mesmo quando há um arquivo que não será usado para algum tipo de *media*, este é baixado de qualquer maneira. Por exemplo, um iPad vai baixar o arquivo com `(max-width:700px)`. Sobre esta questão, existem prós e contras.

Prós em se usar 1 arquivo:

- Somente 1 requisição HTTP
- Mais difícil de esquecer de atualizar

Contras em se usar 1 arquivo:

- Tamanho do arquivo fica maior
- A medida que o projeto cresce, a manutenção fica mais difícil
- Uso de soluções JavaScript para funcionar em IE8 pra baixo

Prós em se usar vários arquivos:

- Para *browsers* que não suportam, o arquivo padrão é menor
- Organização/manutenção ficam melhores

Contras em se usar vários arquivos:

- Várias requisições HTTP
- Mais fácil de esquecer de atualizar algo

Como o **desenvolvimento para web** não é feito de tecnologias funcionando em separado e levando em conta que várias requisições HTTP acabam sendo mais lentas que somente um arquivo de tamanho maior, a recomendação é que seja usada **somente uma folha de estilos**. Vale lembrar que não há uma resposta única, apenas uma recomendação. Sempre vai depender muito da sua situação.

### Media Queries "sobrepostas" ou "empilhadas"

Apesar de não serem termos oficiais dentro do web design responsivo, "sobrepostas" e "empilhadas" traduzem bem as possibilidades e situações que podem acontecer quando se está desenvolvimento sob a filosofia do *responsive web design*. Então, independentemente de se usar um ou vários arquivos de estilo, fato é que é possível, através dos *breakpoints*, **sobrepor** ou **empilhar** as regras usadas.

Veja este exemplo:

```
@media all and (min-width:500px) {
    body {
        background: blue;
```

```
        font-family: serif;
    }
}

@media all and (min-width:700px) {
    body {
        background: red;
        color: #FFF;
    }
}
```

Perceba que as especificações de *media* não são mutuamente exclusivas - quer dizer, se o *viewport* no momento do acesso for 800px, ambas as regras entrarão em ação - e, se isso for feito dessa maneira sem a devida consciência de qual efeito se quer obter, o que se vai conseguir, no final, é muita dor de cabeça e horas a mais de manutenção do arquivo... Não há problema em fazer isso, desde que esta prática seja algo consciente!

Também não é porque agora se está usando Media Queries no arquivo que as regras mais elementares de CSS deixam de valer. Então, a **cascata** e a **especi□cidade** ainda estão presentes! Supondo um acesso com *viewport* de 800px, num código como esse, a página será renderizada com fundo azul ao invés de vermelho:

```
@media all and (min-width:700px) {
    body {
        background: red;
        color: white;
    }
}

@media all and (min-width:500px) {
    body {
        background: blue;
        font-family: serif;
    }
}
```

Para fins de organização, manutenção e facilidade de entendimento da lógica dos estilos que compõem o projeto, geralmente é mais fácil "empilhar" os *breakpoints* e separar as regras devidamente, como em:

```
@media all and (min-width:500px) and (max-width:699px) {
    body {
        background: blue;
        font-family: serif;
    }
}

@media all and (min-width:700px) {
    body {
        background: red;
        color: white;
        font-family: serif;
    }
}
```

Veja que, para que seja aplicada uma família de fonte serifada (`font-family: serif`), foi preciso repetir o par propriedade/valor, já que as regras para um *breakpoint* específico não entram em ação para outro. Para as regras genéricas, que devem entrar em ação sempre, o recomendado é deixar "fora" de um declaração de *media*.

Prós da sobreposição:

- Tamanho de arquivo menor (não é preciso repetir regras)
- Fácil de atualizar regras compartilhada (já que são declaras só uma vez)

Contras da sobreposição:

- Tamanhos de arquivos maiores
- Imagens substituídas/escondidas em partes posteriores do código continuam sendo baixadas por navegadores baseados em WebKit

Prós do empilhamento:

- Browsers somente baixam imagens condizentes ao *breakpoint* atual

Contras do empilhamento:

- Arquivos maiores, já que algumas regras são repetidas em vários *breakpoints*
- Em atualizações, é mais fácil de esquecer de atualizar regras repetidas em alguns *breakpoints*

Sobrepor estilos pode deixar o código mais enxuto, mas a manutenção e entendimento do código ficam mais complicados. Empilhar deixa o código maior, mas mais simples (embora algumas vezes repetido). Quando os estilos entre os *breakpoints* são bastante diferentes (site muito diferente para *mobile* do que para *desktop*, por exemplo), é aconselhado empilhar; do contrário (casos mais comuns), sobrepor.

### Versões antigas do IE: Comentários Condicionais ou JavaScript?

Para versões do Internet Explorer anteriores à 9, não há suporte para Media Queries (por essa você já esperava!). Para a maioria dos projetos no Brasil, ainda é importante dar suporte para, pelo menos, IE7+. É preciso então fazer isso de alguma maneira. Para tanto, é possível usar Comentários Condicionais ou alguma solução JavaScript.

Em relação aos Comentários Condicionais, no próprio blog da Microsoft é possível encontrar soluções para especificar estilos para Windows Phone (http://ow.ly/cTZ9i). Um exemplo seria:

```
<link rel="stylesheet" media="all" href="global.css">
<link rel="stylesheet" media="all and (max-width: 700px)"
                       href="mobile.css">
<!--[if IEMobile 7]>
    <link rel="stylesheet" media="all" href="mobile.css">
<![endif]-->
```

Ou:

```
<link rel="stylesheet" media="all" href="global.css">
<link rel="stylesheet" media="all and (min-width: 700px)"
                       href="desktop.css">

<!--[if (lt IE 9)&(!IEMobile 7)]>
    <link rel="stylesheet" media="all" href="desktop.css">
<![endif]-->
```

Quanto a soluções JavaScript, existem muitas e muitas, mas algumas das mais conhecidas e usadas são:

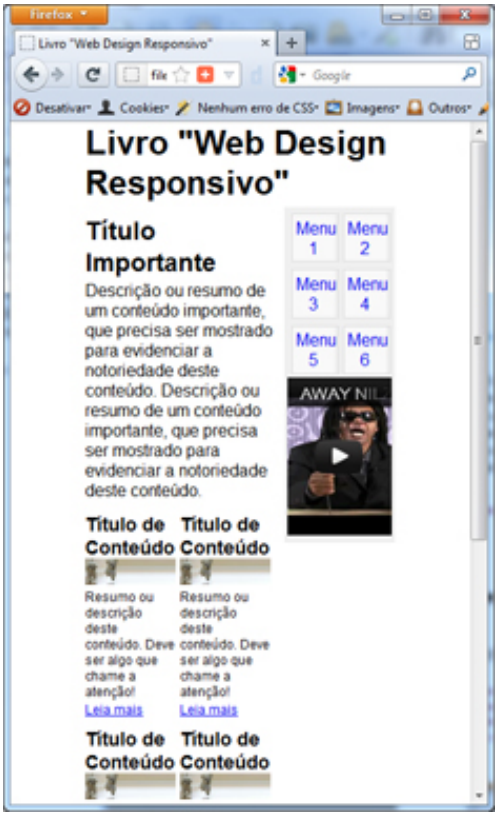
- **Respond**. Suporte somente os parâmetros `min-width` and `max-width`, mas é bastante eficiente no que se propõe.

- **css3-mediaqueries.js**. É mais pesado e mais lento que o Respond, mas suporta mais parâmetros de Media Queries.

Não se esqueç

## 5.8 Media Queries na prática

Vamos à aplicação prática de alguns dos conceitos de Media Queries vistos. Como você deve se lembrar, nosso site de exemplo já conta com muitas características de um site responsivo, tendo imagens (e vídeos) flexíveis e a maioria das áreas de conteúdo possuem largura variável, conforme a resolução da *viewport*. Entretanto, ainda não há aplicação de Media Queries, então o site fica "espremido" quando visto em determinadas resoluções.



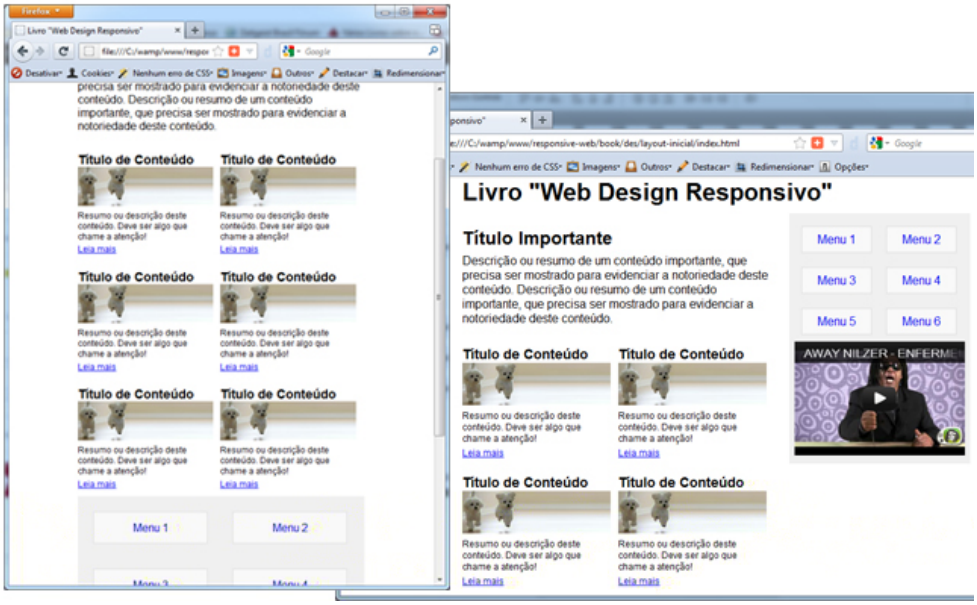

Se nada fosse feito, ainda assim seria uma solução parcialmente aceitável, já que o conteúdo ainda seria completamente acessível, independente da resolução. Entretanto, como já foi mostrado, através de Media Queries, nós podemos planejar melhor como será a experiência dos visitantes, alterando (ou exibindo; ou ocultando) a disposição dos elementos, conforme a experiência que se queira oferecer.

Por exemplo, quando a resolução do navegador no momento da visita for de lar-

gura de até 640px, o site não será apresentado em colunas, mas sim com o conteúdo principal vindo em primeiro lugar (lógico) e a sidebar embaixo. Para tanto, vamos acrescentar algumas linhas de CSS ao final do atual arquivo do site de exemplo, criando um *breakpoint* que contemple resoluções de até `640px`:

```
@media all and (max-width: 640px) {
    .content-main {
        float :none;
    }

    .content-sidebar {
        float: none;
    }
}
```

Com isso, sempre que a resolução do navegador do dipositivo usado for menor ou igual a 640px, a divisão de "coluna principal" e "coluna secundária" no site não mais existirá, sendo apresentado numa só sequência.




Na verdade, o fato de o menu de navegação principal estar sendo apresentado em duas colunas pode ser um contratempo para quem acessa usando uma resolução

bem pequena. Vamos resolver, inserindo o seguinte trecho de código antes da media queria mostrada anteriormente:

```
@media all and (max-width: 320px) {
  .main-nav li {
    float: none;
    width: auto;
    }
}
```

Código que garantirá a seguinte renderização:

Pensando bem, talvez o fato de as notícias serem apresentadas em duas colunas também não contribua para uma boa experiência dos visitantes. Vamos alterar isso no *breakpoint* para resolução de até `640px`, que ficará assim:

```
@media all and (max-width: 640px) {
    .content-main {
        float: none;
    }

        .last-content-call {
            float: none;
            width: auto;
        }

    .content-sidebar {
        float: none;
    }
}
```

O que resultará em:


Como você verá depois que fizer alguns projetos e praticar bastante, o mais difícil não é codificar sites responsivos. O verdadeiro desafio é fazer um bom planejamento

de cada um dos sites para diferentes *ranges* de resoluções para garantir que, em cada um deles, o visitante tenha uma boa experiência de uso, independentemente de qual dispositivo esteja usando.

Será que a melhor forma de vencer esse desafio é encarar o *mobile* logo de cara?

Capítulo 6

# Tópicos de Web Mobile

O caminho traçado através das técnicas para se fazer um bom **web design responsivo** foi emocionante! Vimos muitas das estatísticas que provam que o desenvolvimento *mobile* não é uma mera tendência, como sugerem alguns. Depois, vimos os três pilares do desenvolvimento *mobile*: Layout fluído, imagens (e outros recursos) flexíveis e Media Queries. Conhecendo e sabendo usar esta trinca poderosa, certamente você está preparado tecnicamente para lidar com web design responsivo.

Mas, como foi alertado no fim do capítulo sobre Media Queries, o desafio maior não é dominar a parte técnica do web design responsivo; o maior desafio é saber "como", "quando", "se", "onde" e saber explicar todos os "porques" de usar uma ou outra técnica responsiva.

Uma abordagem importante para se obter o web design responsivo é o **Mobile First**! Mobile First (ou *Dispositivo Móvel Primeiro*) é uma metodologia de desenvolvimento web que apregoa que deve-se **primeiro planejar o design para dispositivos móveis** e, só depois, ir "aumentando" os possíveis dispositivos, até se chegar ao *desktop*. Em termos mais simples, **do menor para o maior** dispositivo.

## 6.1 O que é Mobile First?

*Mobile First* (ou "Móvel Primeiro") é uma metodologia/filosofia/estratégia criada por **Luke Wroblewski** (http://ow.ly/dPqs2) que apregoa que, no desenvolvimento em que o *design* responsivo é levado em conta (ou seja, todo e qualquer projeto *web* seu a partir da leitura deste Livro), deve-se **primeiro planejar para dispositivos móveis** - e, somente depois, projetar, gradualmente, para dispositivos maiores. Em termos mais simples: *do menor para o maior*.

Atualmente, já temos muita diversidade de dispositivos, desde os móveis (*phones*, *smartphones*, *tablets*), passando pelos especializados (*eReaders*, *TVs*, etc), até os mais tradicionais (*desktops*, *laptops*, *netbooks*). E um dos desafios do desenvolvimento *web* moderno (e profissional) é dar suporte a todos estes (e os que ainda estão por vir).

E é por isso que a abordagem *Mobile First* é tão eficiente: começar com *mobile* e projetar com melhoramento progressivo (*progressive enhancement*) permite abranger todos os *devices*. Qualquer dispositivo com acesso a *web* será capaz de acessar o *site* e ter uma experiência funcional. Em seguida, usando deteção de recurso, carregamento condicional de *scripts*, *Media Queries*5 e outras técnicas, é possível permitir que a experiência seja melhorada e otimizada para o contexto do dispositivo que está sendo usado no momento do acesso.

Projetar levando em conta *Mobile First* requer uma revisão profunda e fundamental de um site e, mais importante, requer uma **revisão mental**. Não se trata de uma solução rápida e milagrosa; pelo contrário, *Mobile First* requer um planejamento cuidadoso, tempo e execução séria e com disciplina - o que, você já deve ter imaginado depois das últimas palavras, é algo difícil. Pode parecer assustador no começo, mas, depois que você internaliza a filosofia do *Mobile First*, as recompensas adquiridas são enormes! Ao invés de ter que criar interações totalmente novas toda vez que um novo dispositivo sai, basta otimizar a experiência para o novo contexto sem ter que reinventar a roda (e virar madrugadas) para isso.

## 6.2 Por que Mobile First?

É absolutamente comum que aconteça um profundo e quase sintomático estranhamento quando se tem contato com a abordagem do *Mobile First*. Afinal, antes dessa história toda de web design responsivo, o jeito "normal" de se elaborar designs para web era começar um wireframe de 960px. Só a partir daí, quando o design já estivesse pronto, começaríamos a pensar em outros pontos do projeto.

Depois que se teve contato com o *responsive design*, o próximo passo do projeto seria "reduzir" as propostas de *layout* quando o de 960px estivesse pronto. Por que então fazer o contrário? Por que pensar em focar o desenvolvimento para telas pequenas, touch e outras resitrições antes mesmo da "versão desktop"? Há várias vantagens, tais como a que Luke Wroblewski mostra em seu excelente livro (http://ow.ly/dmxcd), que são:

- Estar preparado para o crescimento explosivo e novas oportunidades *mobile* emergentes (85% dos telefones vendidos em 2011 vieram com browser)
- Obriga você a se concentrar e focar nas principais características de seus produtos/sistemas (o que fazer quando se perde 80% do espaço em tela?)
- Permite-lhe proporcionar experiências inovadoras através de novas tecnologias (geolocalização, eventos de toque, etc)

A principal talvez seja o fato de a abordagem Mobile First forçá-lo a **focar** no que é mais importante e essencial dentre as funcionalidades de seu projeto web. Afinal, quando se trata de um acesso feito através de um dispositivo móvel, as condições de uso e necessidades são bem diferentes do acesso feito através de outros tipos de dispositivos. Não que seja necessário extirpar os conteúdos e funcionalidades que serão apresentados num desktop, mas estaremos dando prioridade ao mais difícil, ao mais limitado.

Vamos passar por alguns pontos para poder enxergar que a abordagem Mobile First é muitas vezes a mais eficiente para um site responsivo.

## 6.3 Notas gerais sobre Mobile First

De maneira geral, as pessoas que usam dispositivos móveis ficam com "um olho no gato e outro no peixe": estão realizando outras atividades enquanto usam seus *devices*. Isso diz muita coisa sobre as diferenças de planejamento para *mobile* e, em relação ao **foco** e **objetividade** que seu *site* precisa, isso diz tudo!

Lembre-se dos perfis de uso *mobile*: Pesquisar, Explorar/Divertir, Check in/Status, Editar/Criar. Esses perfis fazem parte de seu ponto de partida para a definição de sua estratégia mobile, baseado em qual destes seu público-alvo primário se encaixa e o que seu *site*/sistema deve proporcionar.

3 dos pontos mais importantes em relação a *sites* para dispositivos móveis são: **foco no conteúdo**, **foco no conteúdo** e **foco no conteúdo**. Se conseguir atender aos

3, praticamente metade de seu trabalho já estará feito! Lembre-se que as pessoas precisam conseguir acessar outras áreas de seu site de maneira fácil e intuitiva; portanto, seja qual for o perfil que se enquadre em seu público, prover um sistema de navegação igualmente focado e intuitivo é sempre uma boa pedida.

Em se tratando de experiências móveis, a performance ganha um peso ainda maior, já que as pessoas esperam que sites *mobile* sejam tão rápidos quanto sites vistos no *desktop* (ou mais!). A performance e otimização na *web*, em especial no contexto móvel, é fator decisivo de sucesso/crescimento de um negócio *online*. Em seu artigo "14 fatos sobre performance Web e otimizações" (http://ow.ly/eoCoh), **Sérgio Lopes** lembrou bem de alguns fatos a esse respeito:

- Para a Amazon, 1s a mais no carregamento da página custa 1,6 bilhão de dólares por ano;
- O tráfego do Yahoo! aumenta 9% a cada 400ms de melhora na velocidade;
- Ao cortar 2,2s da *landing page* do Firefox, a Mozilla aumentou o número de *downloads* em 15%;
- Um experimento do Google aumentou o carregamento de 0,4s para 0,9s. O tráfego de buscas caiu 20%;
- A Microsoft mostrou que 2s a mais de latência no Bing diminuíam o faturamento em 4,3%;
- Uma diminuição de 6s para 1,2s na página do Shopzilla aumentou as vendas em 12%;
- 50% dos usuários *mobile* abandonam um *site* se ele não abre em até 10s. E 3 em cada 5 não voltam mais;
- Desde 2010, o Google considera a velocidade de carregamento dos sites no *ranking* de buscas.

Por último, caso você ainda esteja pensando que "ganhará" seus visitantes apostando todas suas fichas em *design*, então você ainda não entendeu muita coisa sobre projetar para *mobile*: (na verdade, ainda tem muito a aprender sobre a *web*, em si). É evidente que web design é imprescindível e um bom trabalho neste campo do desenvolvimento web sempre garantiu uma melhora na experiência do usuário e uso do site. Entretanto, não é algo a ser considerado primariamente.

Excetuando-se raríssimos casos, *web sites* não são feitos de design; são feitos de **conteúdo**! Por isso, aposte na *simplicidade* de seu *design* quando projetar para dispositivos móveis. É sério!

## 6.4 Como as pessoas usam dispositivos móveis?

### Perfis de uso

Não é preciso ser um grande observador para saber que as pessoas utilizam dispositivos móveis de maneiras diferentes das que usam dispositivos mais tradicionais. Mas, como estamos tratando de conhecimentos técnicos específicos neste livro, o mero fato de saber que este comportamento é diferente não contribui muito... É preciso mais detalhes sobre isso, um conhecimento mais aprofundado sobre como as pessoas usam estes dispositivos e, a partir disso, projetar e planejar melhor a experiência *mobile* para cada um dos projetos web que você vá fazer parte.

Saiba então que existem determinados **perfis** quando o assunto é comportamento em dispositivos móveis. Através de uma série de estudos e observações, foram encontrados três principais perfis de uso nos dispositivos móveis:

- **Pesquisar (urgente, local).** Pessoas que precisam de uma informação, seja ela qual form, de maneira urgente (informação esta frequentemente relacionada à localização física atual da pessoa). Por exemplo: "Do ponto em que estou agora, onde posso encontrar farmácias próximas?".

- **Explorar / Divertir (entediado, local).** Pessoas que têm algum tempo ocioso/livre e querem aproveitar para procurar alguma forma de distração ou diversão. Por exemplo: "Enquanto espero nesta fila, vou procurar informações sobre os últimos lançamentos do cinema".

- **Check in / Status (repetição, micro-tarefas).** Para os mais "antenados" e/ou que precisam ficar atualizados caso informações que lhes são importantes sejam alteradas. Exemplo: "Preciso saber imediatamente se o status do meu pedido foi atualizado".

- **Editar / Criar (mudança urgente, micro-tarefas).** Quando é preciso realizar uma tarefa importante que tenha que ser feita o mais rapidamente possível. Por exemplo, ajustar uma informação incorreta que a pessoa tenha publicado em seu blog.

Então, através destes perfis, é possível conhecer melhor e saber como as pessoas fazem uso de seus *devices*. Conhecer estes perfis de uso permite que um melhor planejamento de seu projeto web seja feito, sempre com o objetivo de focar em seu público-alvo para proporcionar uma melhor experiência de uso do site ou sistema web, independentemente do dispositivo no momento do acesso.

Dois exemplos interessantes são os do Basecamp e Flickr que, através de uma série de estudos e análises de estatísticas, conseguiram mapear quais as ações mais importantes, que as pessoas mais frequentemente usam. A partir disso, projetaram a experiência *mobile* de seus sites para focar quase que unicamente nas tarefas que as pessoas precisam e querem realizar.

Decisão acertadíssima pois, como vimos nos perfis de uso no *mobile*, o público dos serviços mostrados como exemplo precisam, quando em acesso *mobile*, **agilidade** e **foco** nas tarefas que pretendem executar!

### Hora e lugar

E, além destes principais perfis de uso de dispositivos móveis, também foi possível avaliar outras informações importantes, como hora e lugar em que as pessoas costumam usar seus *smartphones* (http://ow.ly/dnNuz):

- 84% usam em casa
- 80% usam em tempos variados de inatividade durante o dia

- 74% usam enquanto esperam em filas ou por algum compromisso
- 69% usam enquanto estão fazendo compras
- 64% usam no trabalho
- 62% usam enquanto estão assistindo TV
- 47% usam enquanto se deslocam

Estatísticas interessantíssimas, já que, para grande parcela dos desenvolvedores que têm contato com web design responsivo e *mobile first* pela primeira vez, desenvolver para dispositivos móveis remete a sites para executivos de empresas, investidores, instituições financeiras e outros.

As estatísticas sobre horários de uso também são de grande importância. Afinal, é mais do que evidente o fato de que as pessoas usam seus diferentes devices de modo diferente e em horários diretentes. Tomando por base uma série de estatísticas fornecidas pelo **Pocket** (http://ow.ly/dnOTo) - serviço web que permite salvar artigos do interesse para que possam ser lidos posterioremente - em dispositivos Apple, é possível ter uma boa noção do que se está querendo dizer.

No *smartphone*, por exemplo, temos picos em:

- **6am.** Bem cedo, na hora do café da manhã.
- **9am.** Início do expediente para muitas pessoas.
- **5pm - 6pm.** Fim do expediente; volta para a casa.
- **8pm - 10pm.** Horário Nobre (no sofá); hora de ir para a cama.

Já os maiores picos do *tablet* acontecem em horários considerados mais descontraídos e sem maiores compromissos e afazeres. **8pm - 10pm**, depois de um dia de trabalho inteiro, provavelmente já se tomou banho, o jantar já foi servido e as crianças estão distraídas. Em suma, é um momento mais tranquilo.

Com base nas informações sobre hora e lugar em que as pessoas mais usam seus dispositivos móveis, é possível fazer um planejamento mais acurado sobre a estratégia do produto ou serviço *online* que se queira criar ou aprimorar. Pense no seguinte: quais seriam os horários mais indicados para atualizações de gráficos e notícias para um público-alvo de investidores? Ou: qual seriam os melhores horários para atualizações naquele seu super aplicativo de fofocas sobre celebridades?

## 6.5 O conteúdo é o rei

Certamente você já deve ter escutado ou lido a célebre frase "O conteúdo é o rei" - mesmo antes de ter contato com essa história toda de *web design responsivo* e *mobile first* - que, de tanto repetida e divulgada na internet, já não se conhece sua autoria. De nada adiantaria o site mais bonito e usável do mundo se o conteúdo fosse inútil, vazio e/ou confuso.

Quando o assunto é desenvolvimento web pensando em dispositivos móveis, é preciso ter atenção redobrada quanto a isso. Mais do que nunca, agora é preciso dar ênfase total ao conteúdo e tarefas mais comuns que os frequentadores do site ou sistema fazem / podem querer fazer. Quando o acesso é móvel, não há espaço para firulas, elementos que não agreguem à experiência de uso, menus inúteis ou qualquer outros tipos de complicadores!

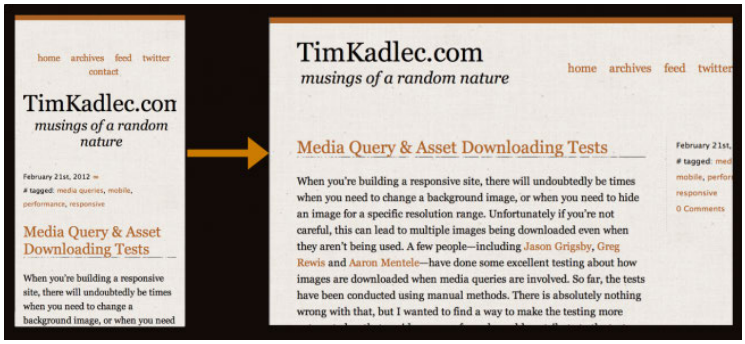
Na verdade, esta máxima deve ser levada em conta no desenvolvimento de qualquer site, mas, em *mobile*, esteja realmente atento a isso! Seus visitantes de dispositivos mobile realmente vão querer ver o que seu site tem a oferecer nos primeiros segundos de sua visita e, caso encontrem algo diferente do conteúdo que estão procurando/querendo, tenha certeza de que a Taxa de Rejeição de seu site vai aumentar.

## 6.6 Padrões de navegação mobile

Tudo bem, a esta altura você já deve ter internalizado a importância maior do conteúdo em seus sites. Depois disso, é preciso garantir que o esquema de navegação também esteja bem projetado, pronto par atender às necessidades dos visitantes prontamente, de maneira eficiente, agradável e não dificultosa.

Apesar de o web design responsivo ser algo relativamente novo, alguns padrões de navegação já foram sistematizados. **Brad Frost** (http://ow.ly/dy56A) fez um ótimo trabalho em organizar a maioria deles, elencando prós e contras de cada abordagem. Organização esta apresentada a seguir de maneira resumida.

### Top nav




Uma das soluções mais fáceis de implementar para a navegação é simplesmente mantê-la no topo. Por causa de sua facilidade de implementação, essa abordagem é encontrada em muitos (talvez a maioria) dos sites responsivos.

#### Prós

- Fácil de implementar
- Não é preciso JavaScript
- Sem necessidade de "malabarismos" CSS

#### Contras

- Pode ocasionar problemas de altura

- Não escalável
- Pode ocasionar problemas com os links muito próximos

## Âncora no rodapé


Esta solução mantém a lista de navegação no rodapé do site enquanto o cabeçalho contém um link de âncora simples apontando para a navegação no rodapé.

### Prós

- Fácil de implementar
- Não é preciso JavaScript
- Sem necessidade de "malabarismos" CSS
- Único botão no cabeçalho

### Contras

- A navegação por âncora pode desorientar algumas pessoas
- Alguns podem considerar a solução "não elegante"

### Menu de seleção


Uma maneira de lidar com menus de navegação é transformá-los numa lista de links em um menu de seleção para telas menores. Isso evita os problemas da abordagem Top nav e é uma maneira inteligente de economizar espaço em tela.

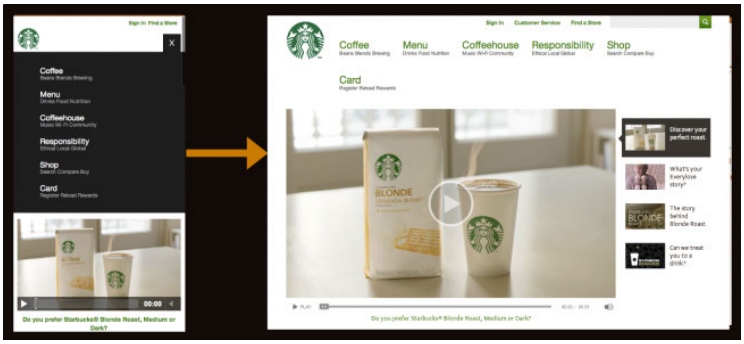
#### Prós

- Não ocupa espaço em tela
- Mantém toda a interação no cabeçalho
- Facilmente reconhecível

#### Contras

- Difícil de estilizar elementos `select`
- Potencialmente confuso
- Submenus podem parecer “estranhos”
- Necessita de JavaScript

## Alternância




A abordagem Alternância é semelhante à abordagem Âncora no rodapé, mas, ao invés de saltar para baixo para uma âncora na parte inferior da página, o menu se abre no próprio local. É uma abordagem de boa aparência e é relativamente fácil de implementar.

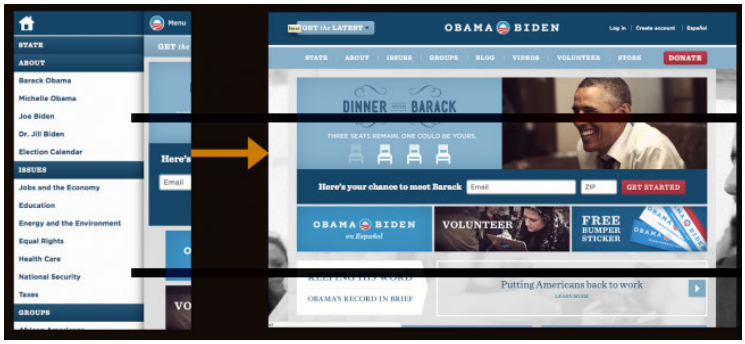
### Prós

- Mantém a interação num só local
- Elegante
- Facilmente escalável

### Contras

- Performance (ao realizar a animação do slide)
- Precisa de JavaScript

### Slide à esquerda




O menu de navegação é acessado por um ícone do menu que, quando acessado, exibe as opções do menu através de uma animação de *slide* que começa à esquerda, movendo o conteúdo principal para a direita.

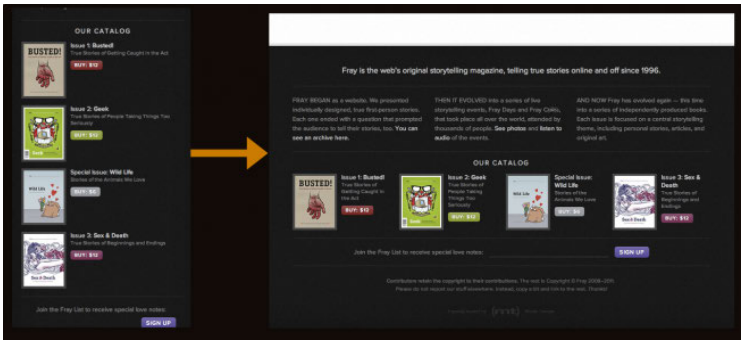
#### Prós

- Não ocupa espaço em tela
- Mantém a interação num só local
- Boa aparência

#### Contras

- Técnica relativamente avançada
- Não escala muito bem
- Potencialmente confuso

### Somente no rodapé




A navegação Somente no rodapé é semelhante à abordagem Top nav, só que sem a âncora no cabeçalho.

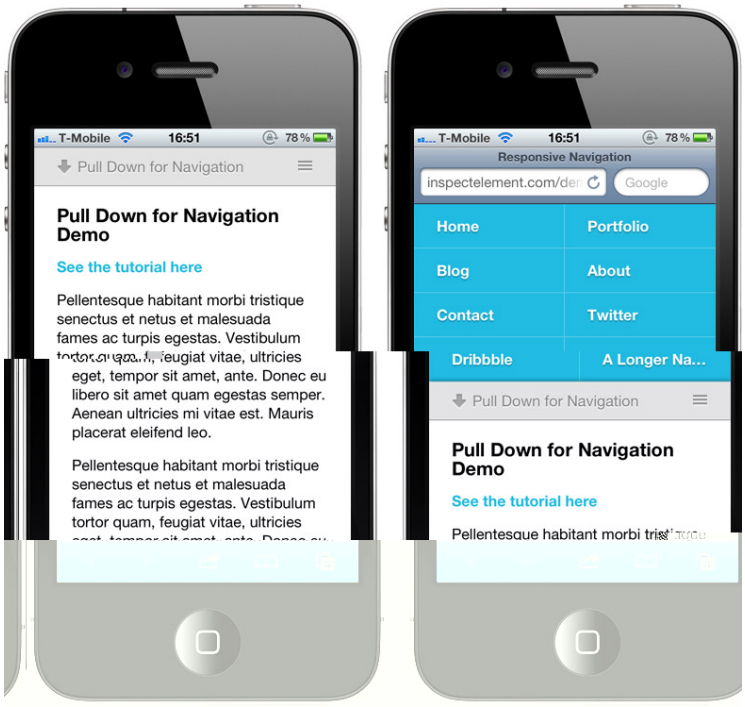
#### Prós

- Não ocupa espaço no cabeçalho

#### Contras

- Difícil de encontrar e acessar o menu (deve-se percorrer todo o conteúdo)

### Navegação Pull Down




Revela o menu de navegação num movimento de slide que empurra o restante do conteúdo para baixo.

#### Prós

- *Sexy*! ;-)
- O movimento de slide *top-down* já é uma convenção em *smartphones*

#### Contras

- Técnica relativamente avançada
- Precisa de instrução de uso (geralmente, um *label*)

### Considerações sobre padrões de navegação mobile

Foram apresentadas algumas das abordagens mais comuns quando falamos em sistemas de navegação para sites que se valem de **web design responsivo**. Certamente, não foram apresentadas todas e, como o assunto é novo, sugestões de melhorias e novas abordagens serão desenvolvidas, propostas e discutidas. O importante é tentar chegar o mais próximo possível do objetivo de aliar navegação acessível, espaço em tela e facilidade de uso (o que não é uma tarefa simples).

Caso esteja interessado em como pode implementar cada uma dessas abordagens, o artigo *Responsive Menus: Enhancing Navigation On Mobile Websites* (http://ow.ly/dyb4x) é o mais indicado.

## 6.7 10 princípios de design para interfaces mobile

O consultor de *mobile* **Jonathan Stark** (e este nome, por si só, já diz muito sobre sua capacidade) elaborou *10 princípios de design de interface móvel* (http://ow.ly/dJnk8) que são bastante conhecidos. Vamos passar por eles de forma sucinta.

### 1. Mentalidade móvel

Devido às diferenças entre *mobile* e *desktop*, é imperativo que você faça uma imersão na mentalidade *mobile* antes de começar.

**Seja focado.** Mais não é melhor. Edite suas *features* impiedosamente. Você vai ter que deixar algumas coisas para fora.

**Seja único.** Saiba o que faz com que seu aplicativo seja diferente e amplifique isso. Há muitos peixes no mar de aplicativos móveis. Se não há nada de especial em sua aplicação, por que alguém iria querê-la?

**Seja encantador.** Dispositivos móveis são intensamente pessoais. Eles são nossos companheiros constantes. Aplicativos que são amigáveis, confiáveis e divertidos são uma delícia de usar e as pessoas vão se tornar muito ligadas à experiência.

**Seja atencioso.** Desenvolvedores de *mobile* muitas vezes se concentram no que seria divertido desenvolver, no seu próprio modelo mental do aplicativo ou seus objetivos pessoais. Estes são bons lugares para começar, mas você tem que se colocar no lugar de seus usuários se você espera criar experiências envolventes.

### 2. Contexto móvel

A imagem do profissional ocupado, correndo pelo aeroporto com um saco na

mão e smartphones no outro é o que muitas pessoas têm em mente quando se pensa sobre o contexto móvel. É certamente um contexto, mas não é o único. Para começar a colocarmo-nos na pele de nossos usuários, temos de considerar três grandes contextos móveis: Entediado, Ocupado e Perdido.

**Entediado.** Há muitas pessoas usando seus *smartphones* no sofá de casa. Neste contexto, as experiências imersivas e deliciosas, voltadas para uma sessão de maior tempo de uso, são um grande possibilidade. Ainda assim, interrupções são altamente prováveis, então, tenha certeza de que seu aplicativo pode continuar de onde o usuário parou. Exemplos: Facebook, Twitter, Angry Birds, navegador web.

**Ocupado.** Este é o cenário "atravessando o aeroporto". A capacidade de realizar micro-tarefas de forma rápida e confiável com uma mão em um ambiente agitado é crítica. Lembre-se de que o usuário terá a "visão de túnel" neste contexto, então objetivos claros e design arrojado são importantes. Exemplos: TripIt, e-mail, calendário, internet banking.

**Perdido.** Os usuários que estão no trânsito, em ambientes desconhecidos ou em ambientes familiares, mas interessados em algo ao redor, se enquadram nessa categoria. Neste contexto, a conectividade e a bateria são grandes preocupações, então você deve oferecer algum nível de suporte offline e ser econômico com a utilização de geolocalização e outros gargalos de bateria. Exemplos: Mapas, Yelp, Foursquare.

Para maiores informações, dados e estatísticas a esse respeito, relembre sobre os perfis de uso de dispositivos móveis.

## 3. Orientações gerais

Diferentes aplicações exigem diferentes abordagens, design e técnicas. Dito isto, a natureza inerente de um dispositivo *touchscreen* de bolso sugere várias diretrizes globais, ou seja, as coisas que sempre importam.

**Responsividade.** Capacidade de resposta é absolutamente crítico. Se o usuário faz algo, sua aplicação deve reconhecer a interação instantaneamente. Note-se que a capacidade de resposta e velocidade não são a mesma coisa. Tudo bem se certas operações levarem tempo. Apenas certifique-se que o usuário saiba que as coisas estão funcionando/acontecendo.

**Polidez.** Polidez é extremamente valiosa. Em virtude da natureza do "companheiro constante" da nossa relação com os *smartphones*, prestar muita atenção nos pequenos detalhes, isso será notado e apreciado.

**Dedos.** Com o advento das interfaces *touchscreen*, todo mundo está sempre falando de "toque nisso" e "toque naquilo". Na realidade, é para o polegar que preci-

samos projetar. A menos que o usuário esteja interagindo com seu *smartphone* com as duas mãos, é quase impossível a interação com outro dedo qualquer e, mesmo quando se mexe com as duas mãos, o mais provável é digitar com os dois polegares. Polegares são o padrão.

**Alvos.** Devido a uma série de estudos sobre o tamanho e forma do polegar, parece que o número mágico para os elementos de UI amigáveis ao polegar é de 44 pixels. Existem muitas exceções, mas esta é uma regra geral. Você também deve estar consciente sobre onde onde colocar um *targets* em relação aos outros (caso hajam). Por exemplo, colocar o botão "Apagar" (*backspace*) diretamente ao lado do botão de envio em um aplicativo SMS seria uma má idéia.

**Conteúdo.** A revolução das interfaces de toque é que elas nos permitem interagir diretamente com nosso conteúdo. Isso remove abstrações (como *mouse* e *trackpad*) e está mais de acordo com a forma como nossos cérebros funcionam. Uma criança de 2 anos consegue usar um *tablet* sem dificuldade, mas um *laptop* já é um mistério. Aproveite o poder intuitivo da interface de toque, minimizando "dificultadores" (botões, abas, caixas de seleção, sliders e assim por diante) sempre que possível e colocando seu conteúdo sempre em foco.

**Controles.** Quando é preciso adicionar controles, tente colocá-los na parte inferior da tela (em outras palavras, sob o conteúdo). Pense em uma máquina de somar, uma balança de banheiro ou até mesmo um computador: os controles estão abaixo do visor. Se não fosse, não seríamos capazes de ver o que estava acontecendo com o conteúdo enquanto se dá o uso.

**Rolagem.** Evite a rolagem. Ter uma tela sem rolagem causa uma melhor "sensação" do que uma tela com rolagem. Claro, certas telas devem ser projetadas para rolar, mas é bom evitá-las sempre que possível.

## 4. Modelos de navegação

Há uma abundância de modelos de navegação inovadores para aplicativos móveis, mas, se você estiver tendendo a usar um dos modelos comuns de navegação, certifique-se de escolher o que faz mais sentido em sua aplicação.

Mais a esse respeito, leia novamente sobre padrões de navegação *mobile* na seção
.

## 5. Inputs do usuário

Digitação é complicado até mesmo nos melhores dispositivos, então, você deve

fazer o que puder para tornar esse tipo interação mais fácil para os usuários. Por exemplo:

- Há cerca de uma dúzia de variações de teclado em smartphones populares (texto, número, e-mail, URL e assim por diante). Considere cada um dos seus campos de entrada e certifique-se de exibir o teclado que será mais útil para a entrada de dados que está sendo feita.
- Corretores ortográficos podem ser tão divertidos, quanto frustrantes. Considerar cada um dos seus campos de entrada e decidir quais campos devem estar ativos (como auto-correção, auto-capitalização e *autocomplete*).
- Se seu aplicativo precisa de um monte de digitação, você deve garantir que o teclado no sentido horizontal (*landscape*) esteja preparado para dedos "gordinhos".

### 6. Gestos

Um dos aspectos mais emblemáticos de interfaces de toque modernas é que elas suportam gestos para interações. Gestos são realmente muito legais e úteis, mas há várias coisas que você precisa manter em consideração:

**Invisibilidade.** Gestos são invisíveis, por isso sua descoberta é um problema. Você tem que decidir como revelar sua existência para o usuário. Uma das abordagens mais inteligentes é como na dos iPads promocionais expostos em lojas de varejo da Apple. Quando uma página é carregada primeiro, as áreas de rolagem fazer um rápido "*scroll* reverso" até sua posição padrão. Isto imediatamente convida a um gesto súbito do usuário sem ter que indicar, explicitamente, quais são as áreas de rolagem.

**Duas mãos.** Gestos *multi-touch* requerem o uso das duas mãos. Isso é particularmente evidente em aplicativos de mapas do iOS que usam o gesto de "abrir pinça" para diminuir o zoom. Mas, se, por exemplo, você está viajando em uma cidade estrangeira com um café numa mão e o telefone na outra, essa é uma limitação irritante. O Android tem uma abordagem melhor sobre esta questão, incluindo botões de "zoom in" e "zoom out" próximos ao mapa.

**Bom de ter.** Na maioria dos casos, gestos podem ser considerados como "bom te ter", mas não críticos. É como alguns tipos de atalhos de teclado: usuários avançados vão amá-los, mas a maioria das pessoas sequer sabem que estão lá.

**Sem substituição.** Ainda não existe um padrão comum para gestos, por isso é muito cedo para a maioria dos aplicativos descartarem controles visíveis que podem ser usados com um único dedo.

## 7. Orientação

A orientação *portrait* é, de longe, a mais popular. Por isso, otimize primeiro para este caso. Se seu aplicativo precisa de muita digitação, você deve dar suporte a orientação landscape para as pessoas poderem acessar o teclado maior.

Quando a orientação muda inesperadamente, isso é desorientador. Se você acha que seu aplicativo será usado por longos períodos de tempo (por exemplo, o Kindle Reader), considere a adição de uma trava de orientação diretamente no aplicativo.

## 8. Comunicações

**Forneça feedback.** Forneça *feedback* imediato para cada interação. Se você não fizer isso, o usuário não vai saber se o aplicativo travou ou se erraram o alvo/interação que tentaram interagir. O *feedback* pode ser tátil ou visual. Se o usuário tiver solicitado uma ação que vai levar um longo tempo, apresente um indicador de *loading* para que saibam que a solicitação foi enviada e está sendo processada.

**Alertas modais.** Alertas modais são extremamente agressivo e invasivos para o fluxo do usuário, assim, você só deve usá-los quando algo estiver seriamente errado. Mesmo assim, tentar atenuar a intensidade, mantendo a linguagem reconfortante e amigável. Lembre-se de não usar alertas modais para mensagens do tipo "Para sua informação".

**Con□rmações.** Quando você tem que pedir a um usuário para confirmar uma ação, é aceitável exibir um diálogo de confirmação modal (como "Tem certeza de que deseja excluir este projeto?"). As confirmações são menos invasivas do que alertas porque elas aparecem em resposta a uma ação do usuário e, portanto, num contexto até esperado. Lembre-se de fazer com que a o botão padrão seja a escolha "mais segura" da caixa de diálogo para ajudar a evitar inadvertidas ações destrutivas.

## 9. Inicialização do aplicativo

Quando um usuário volta para seu aplicativo depois de já o ter usado antes, você deve retomar as operações exatamente onde o usuário parou. Isso dará a "ilusão de velocidade" e contribue para uma sensação geral de capacidade de resposta.

Se possível, a tela de inicialização exibida quando o aplicativo é iniciado (ou aces-

sado) deve ser uma tela desprovida de conteúdo, apresentando somente o logo ou alguma imagem. Qualquer coisa que pareça interativa (como botões, links, ícones, conteúdo) vai criar frustração ao convidar para um interação "falha".

Resista fortemente à tentação de colocar elementos/materiais de *branding* na tela de *launch*. Eles fazem o usuário se sentir como se estivesse vendo um anúncio e eles se "ressentem" com isso, tendendo a ir embora.

### 10. Primeiras impressões

**Seu ícone.** Seu ícone tem de competir por atenção em um mar de outros ícones. Sendo esse o caso, pense nisso mais como um cartão de visitas do que uma obra de arte. Seja literal: mostre o que seu aplicativo faz. Use uma silhueta forte e opte por um mínimo de textos. Um ícone polido sugere um aplicativo polido, então vale a pena dedicar tempo sério e dinheiro para fazê-lo direito.

**Primeiro lançamento**. O primeiro lançamento é uma situação para encantar ou decepcionar. Se um novo usuário fica confuso ou frustrado ao tentar se familiarizar com seu aplicativo, eles vai abandoná-lo o mais rápido que conseguir. Se o aplicativo fornece funcionalidades complexas, você pode querer incluir um "Dicas e Truques" ou algumas telas com orientações gerais. Note que estas dicas não servem para substituir um bom projeto. Se você se encontrar criando um monte de texto de ajuda, isso é um forte indicativo de que a UI de sua aplicação precisa de um retrabalho.

Capítulo 7

# Continuando seus estudos

E eis que você chegou ao último capítulo, são e salvo! Para ajudar nos seus estudos, apresentamos aqui uma compilação de recursos sobre *web design* responsivo.

## 7.1 Artigos/Tutoriais

Ler artigos e tutoriais é a base para se manter atualizado sobre o que acontece no mundo da responsividade. Configura algumas boas referências e continue os estudos!

### Considerations for Mobile Design: Behavior

O artigo "Considerations for Mobile Design: Behavior" (http://ow.ly/eor9U) traz informações de caráter importantíssimo sobre o comportamento no ambiente *mobile*, diferenças entre formas de interação, UIs *mobile* e mais. Leia!

### Convert a Menu to a Dropdown for Small Screens

Nos padrões de navegação *mobile*6.7, você viu que uma das técnicas é transformar o menu numa lista de *links* em um menu de seleção para telas menores. No "Convert a Menu to a Dropdown for Small Screens" (http://ow.ly/dUNa6), você vai saber, passo a passo, como realizar este efeito.

### Design Process In □ e Responsive Age

O artigo "Design Process In The Responsive Age" (http://ow.ly/eokZV) aborda pontos muito interessantes sobre o processo de *design* para *sites* com *web design* responsivo. Vale a pena dar uma olhada, pois o planejamento de um projeto é uma de suas principais e imprescindíveis etapas!

### Hiding and revealing portions of images

Em "Hiding and revealing portions of images" (http://ow.ly/eoprm) você vai aprender técnicas mais avançadas de apresentação de imagens em *web designs* responsivos e vai aprender como ocultar determinadas partes de imagens quando for mais adequado.

### Optimizing your email for mobile devices with the @media query

No tutorial "Optimizing your email for mobile devices with the @media query" (http://ow.ly/dUNmd), você vai aprender como tornar seus e-mails para *mobile* usando *Media Queries*. Afinal, é importante garantir mensagens e *newsletters* responsivos para uma experiência *mobile* completa.

### Responsive Data Tables

No tutorial "Responsive Data Tables" (http://ow.ly/dUN5f), você vai conhecer algumas ideias excelentes sobre como tratar tabelas e dados tabulares em projetos com *web design* responsivo.

### Responsive Work□ow

Em "Responsive Workflow" (http://ow.ly/eoxEK), você vai encontrar a proposta de uma metodologia sobre como planejar, desenvolver e lançar projetos responsivos. É uma leitura interessante que pode ser a base fundamental de seus projetos a partir de agora!

### □ e Mobile Playbook

The Mobile Playbook, Um Guia para Executivos Ocupados Vencerem no Mercado Mobile (http://ow.ly/e0AK3), da Google, é um livro interativo com informações preciosas sobre o mundo de dispositivos móveis, marketing *mobile*, considerações, dicas e estatísticas sobre como vencer no mercado móvel.

## 7.2 Bookmarklets

Um *bookmarklet* é um trecho em JavaScript que é armazenado como uma URL podem ser salvos nos favoritos de qualquer *browser*. É interessante que você tenha alguns também para o desenvolvimento com *web design* responsivo.

### Media Query Bookmarklet

Ao ativar o Media Query Bookmarklet (http://ow.ly/dUO7c), ele analisa os arquivos CSS incluídos da página atual e cria uma série de declarações de *media query* que são mostradas na tela, em tempo real, enquanto você redimensiona a janela do navegador.

### Responsive Design bookmarklet

Responsive Design Bookmarklet (http://ow.ly/eokvR) permite que você crie um *bookmarlet* em tempo real para testes com *web design* responsivo. Defina os tamanhos que gostaria de realizar seus testes e salve seu *bookmarlet* personalizado para uso futuro.

## 7.3 Esboço e planejamento

Já que, a partir de agora, você só trabalha com web design responsivo e Mobile First, seus esboços também devem sofrer uma alteração para acompanhar esta nova fase. Simplesmente escolha um dos mostrados e comece a desenhar!

### A Simple Device Diagram for Responsive Design Planning

A Simple Device Diagram for Responsive Design Planning (http://ow.ly/eogow), como sugere o próprio nome, é um diagrama simples para ser usado no planejamento de *web design* responsivo, mostrando uma "régua" com as principais resoluções dos *devices* mais usados no mundo.

### Responsive Design Sketchbook

Que tal um caderno quadriculado especialmente feito para planejamento de *web designs* responsivos? Esta é, justamente, a proposta do Responsive Design Sketchbook (http://ow.ly/dUMH9). Por um preço acessível, é possível adquirir este caderno espiral com 50 páginas para o seu planejamento.

### Responsive Web Design Sketch Sheets

Responsive Web Design Sketch Sheets (http://ow.ly/dPHlf) é um conjunto de modelos demarcados, indicando vários tamanhos de tela, para o planejamento e esboço das telas em sites responsivos serem bem rápidos.

### UX Sketching And Wireframing Templates For Mobile Projects

UX Sketching And Wireframing Templates For Mobile Projects (http://ow.ly/dSBKo) é um conjunto de 28 folhas para esboços e *wireframes* em PDF para diversas plataformas, dentre as quais Android, BlackBerry, iOS (iPad and iPhone), Meego, Symbian, webOS e Windows Phone 7.

## 7.4 Ferramentas

Agora que você já sabe bastante coisa sobre *web design* responsivo, é importante acrescentar no seu *kit* pessoal algumas ferramentas para ajudar com o desenvolvimento responsivo.

### Fluid Grids

Já foram mostrados vários *frameworks* que possuem *grids* fluídos para desenvolvimento de *sites* responsivos. Mas, e se você quiser ter o seu próprio *grid*? Com Fluid Grids (http://ow.ly/dUOhD) essa tarefa vai ser mais fácil do que você imagina: somente especifique quantas colunas, o tamanho de cada uma e o espaço entre elas e pronto, faça o *download* de seu grid personalizado!

[PXtoEM]

PXtoEM (http://ow.ly/eoj9U) é uma tabela interativa de conversão de pixels para outras unidades como EMs, porcentagem e Pontos. Possui alguns valores pré-definidos e permite a inserção de valores-base para a conversão.

### resizeMyBrowser

No resizeMyBrowser (http://ow.ly/dUNNr), você encontra botões com as resoluções mais comuns (dos *devices* mais populares) que, ao serem clicados, abrem uma nova janela com aquele tamanho escolhido.

### □ e Responsive Calculator

Obviamente, você se lembra da fórmula mágica do *web design* responsivo3.2. Mas, caso tenha alguns problemas/dificuldades com cálculos, The Responsive Calculator (http://ow.ly/eoeDr) é uma ferramenta imprescindível para você, já que se trata de uma espécie de calculadora específica para *web design* responsivo, ajudando a converter *pixels* para porcentagens.

## 7.5 Inspiração

Agora que você já sabe bastante sobre *web design* responsivo, é de interesse obervar alguns projetos e exemplos de *sites* e recursos já desenvolvidos e/ou baseados na responsividade.

### Media Queries

Uma das primeiras e mais conhecidas galerias de *sites* responsivos, no Media Queries (http://ow.ly/eoyev) é possível companhar projetos responsivos de excelente qualidade.

### Mobile UI Patterns

Se você está começando agora (ou, mesmo, já está dentro) do mundo do desenvolvimento *mobile*, vai gostar bastante de conhecer o Mobile UI Patterns (http://ow.ly/eoyyt), que apresenta soluções de UI para *mobile* separadas por categorias.

## 7.6 JavaScript puro

Em alguns casos, soluções em JavaScript puro também são uma ótima pedida para a solução de problemas com *web design* responsivo.

### Adapt.js

Adapt.js (http://ow.ly/dUOSy) é um JavaScript muito leve (menos de 1KB quando minificado) que determina qual arquivo CSS deve ser carregar antes de o navegador processar uma página. Se o navegador é redimensionado, Adapt.js simplesmente verifica a nova largura e provê apenas o CSS necessário, quando necessário.

### foresight.js

foresight.js (http://ow.ly/eorEY) provê a informação que diz se o dispositivo que está fazendo o acesso é capaz de visualizar imagens em alta resolução (como o iPad 3ª geração, por exemplo) antes da imagem ser solicitado no servidor.

### Respond

Respond (http://ow.ly/dPGhQ) é um *poly fill* que faz com que IE6-8 reconheçam *min-width* e *max-width* para o uso de *Media Queries*.

### Retina Images

Retina Images (http://ow.ly/eoduN) automaticamente serve imagens em alta resolução para dispositivos que possuem *retina display*. Seu uso é bastante simples e, para os *sites* que querem se valer deste extra, realmente vale a pena!

## 7.7 Plugins jQuery

É possível, sim, fazer bons *sites* usando somente HTML e CSS (pelo menos no nível atual destas tecnologias), mas fato é que, para proporcionar experiências realmente marcantes e envolventes, um toque de *JavaScript* se faz necessário. Como, não por acaso, jQuery é a biblioteca JS mais usada no mundo, aí vai uma lista de *plugins* relacionados a *web design* responsivo.

### Camera

Camera (http://ow.ly/dPCg6) permite fazer *slideshows* responsivos incríveis. Conta com *loading*, miniaturas, opções, *callbacks* e, seguindo o estilo jQuery de ser, é simplíssimo de usar.

### FitText

FitText (http://ow.ly/dPB4V) torna a propriedade CSS `font-size` flexível. Utilize este *plugin* em seu projeto para conseguir títulos escaláveis, da largura total do elemento-pai.

### Flexslider

Flexslider (http://ow.ly/dU3Yo) é outro excelente *plugin* para *slideshows* responsivos. Funciona com marcação simples, dá suporte a vários efeitos de animação, *sliders* múltiplos, API, callbacks e tudo o mais que um *plugin* completo possui.

### Isotope

Isotope (http://ow.ly/eobNM) é um excelente *plugin* que além de reorganizar elementos na tela (com um efeito bem interessante) em tempo real, conforme o tamanho da janela do navegador, permite usar opções de filtragem para os elementos que estão sendo apresentados.

### scrolldeck

scrolldeck (http://ow.ly/dPEjn) permite fazer *sites* e apresentações com "*decks* rolantes", ou seja, você clica em um *link* e a página rola até o ponto especificado.

### Supersized!

Supersized! (http://ow.ly/dPEHZ) permite *slideshow* de imagens em tela cheia, seja qual for a resolução usada por quem acessa o site.

### TinyNav.js

Nos padrões de navegação *mobile* 6.7 você viu que uma das técnicas é transformar o menu numa lista de *links* em um menu de seleção para telas menores. TinyNav.js (http://ow.ly/eoc1I) serve, justamente, para isso!

### Wookmark

Wookmark (http://ow.ly/eocfE) detecta o tamanho da janela do navegador que está realizando o acesso e, a partir disso, rearranja os elementos da página em colunas. Na prática, é uma alternativa mais "enxuta" ao *plugin* Isotope.

## 7.8 Testes de responsividade

Existem diversas ferramentas pela *web* à fora que auxiliam a fazer os testes de seus *web designs* responsivos, tornando a tarefa de verificar como as coisas estão ficando em diferentes resoluções de maneira mais prática. Confira algumas.

### Demonstrating Responsive Design

Demonstrating Responsive Design (http://ow.ly/dULxj) permite visualizar como um *site* fica visto em um celular em *landscape* e *portrait*, *tablet* também com as 2 orientações e *desktop*, alterando, de forma interativa, as visualizações para visualizar.

### Responsive Play

Responsive Play (http://ow.ly/dSD1b) é uma ferramenta brasuca, desenvolvida por Sérgio Lopes. Ela ajuda bastante a tarefa de checar o desenvolvimento com *web design* responsivo ao permitir visualizar uma animação de um *viewport* se estendendo conforme uma faixa de largura e tempo definidos por quem usa a ferramenta. Bem interessante!

### Responsive Web Design Testing Tool

Com Responsive Web Design Testing Tool (http://ow.ly/dUNFe), é possível visualizar qualquer sitenas larguras de 240, 320, 480, 768, 1024 *pixels*. Bastante útil e economiza um bom tempo por mostrar todas as "versões" simultaneamente.

### Screen□y

A partir de um URL informado, Screenfly (http://ow.ly/eocUU) permite escolher a visualização de um *site* em diversos tipos de dispositivos (*Desktop*, *Tablet*, *Mobile* e TV), inclusive com variações das principais resoluções/modelos de cada um.

### Screenqueri.es

Screenqueri.es (http://ow.ly/eodh6) é uma ferramenta de testes de sites em várias resoluções para os adoradores de *pixel-perfect*. Além de permitir visualizar o *site* em diversas resoluções pré-definidas pelos modelos de aparelhos mais conhecidos, permite arrastar largura e altura para ver como o *site* se apresenta dentro de um *grid* de marcação. Muito bom para testar o site em vários *breakpoints* diferentes.

## 7.9 Templates e Frameworks

Depois de já ter aprendido bem os princípios e técnicas do *web design* responsivo, você pode querer agilizar os projetos usando modelos ou *frameworks* responsivos. Veja os melhores/mais usados no mundo, estude cada um deles e encontre o que melhor se encaixa ao seu estilo.

### 1140 CSS Grid

1140 CSS Grid (http://ow.ly/dPJw1) se encaixa perfeitamente em um monitor de `1280px` e, em monitores menores, torna-se fluído e se adapta à largura do navegador. A partir de certo ponto, ele usa *Media Queries* para servir uma versão móvel que, essencialmente, empilha todas as colunas para que o fluxo de informação ainda faça sentido.

### 320 and Up

320 and Up (http://ow.ly/dPLxU) é um dos modelos para design responsivo mais conhecidos. Além das características mais óbvias (como layout fluído, imagens flexíveis, etc), ele tem a vantagem de agradar aos que gostam de LESS e SASS para mexer com o CSS e já vem com Modernizr e Selectivizr.

### Foundation 3

Foundation 3 (http://ow.ly/dQqdG) se auto intitula *o mais avançada framework responsivo de front-end do mundo*. Seu poderoso sistema de grid é feito com SASS, assim como seus elementos de UI e demais características. Já conta com alguns *add-ons*, como templates HTML, ícones de fontes, tabelas responsivas e mais.

### Frameless

Frameless (http://ow.ly/dPIZS) é um grid feito para designs responsivo com colunas de largura-fixa, ou seja, ao invés de tudo ser fluído, dependendo da resolução mais ou menos colunas são exibidas na tela. Seu *slogan* é "Adaptar coluna por coluna, não pixel por pixel".

### Initializr

Initializr (http://ow.ly/eoAjK) é um gerador de templates para HTML5 para ajudar a começar a iniciar novos projetos. O kit conta com HTML5 Boilerplate em

combinação com Twitter Bootstrap, jQuery e Modernizr. Ele gera um modelo limpo personalizável com apenas o que é preciso para começar o projeto. Altamente recomendado!

### inuitCSS

inuitCSS (http://ow.ly/dPKio) é um *framework* para *design* responsivo que conta com uma série de *plugins* para *breadcrumb*, *dropdown*, navegação, dentre outros.

### Responsive Grid System

Responsive Grid System (http://ow.ly/eoz2M) é uma maneira rápida, fácil e flexível de criar *web sites* responsivos. Seu sistema de *grids* é realmente fácil de entender e usar através de uma marcação simples e rápida.

### Skeleton

Skeleton (http://ow.ly/dPID6) é uma coleção de arquivos CSS e JS que form uma série de componentes de UI para ajudar a desenvolver rapidamente *sites* que são bem vistos em qualquer resolução.

### Twitter Bootstrap

Twitter Bootstrap (http://ow.ly/dPHWq) é um *framework* completo para *front-end*. Possui diversos componentes de UI, *plugins javascript* e um *grid* responsivo que vai agilizar bastante o desenvolvimento de seus projetos futuros.

## 7.10 Palavras finais

Não deve ser nenhuma surpresa que as técnicas e conhecimentos mostrados no Livro são tão dinâmicos e mutáveis quanto a própria *web*! Então, nada de achar que agora você é o Sr. (ou Sra.) Responsividade; você está a apenas começando a jornada e, como diria **Morpheus**, *há uma diferença entre conhecer o caminho e trilhar o caminho*.

Bons estudos!

# Referências Bibliográficas