



Media Queries

Transcrição

MEDIA QUERIES

Já foi discutido sobre a aplicação do design responsivo e como implementar um design fluído eficiente. Porém, o design fluído resolve somente metade do problema de um site responsivo.

DESIGN FLUÍDO RESOLVE SÓ METADE DO PROBLEMA

Porque quando se trabalha com medidas flexíveis, como as porcentagens, os `em` etc., teremos um site que se adapta muito bem para variadas resoluções, à medida que vai se redimensionando, mas como existem muitos tipos de resolução, por exemplo, desde um desktop até um celular pequeno, talvez as adaptações não fiquem tão boas nos extremos, com acontece no caso do exercício para criação de um site com três colunas.

As três colunas flexíveis vão se adaptando a várias resoluções, porém quando se chega em um layout menor pode acontecer o seguinte:

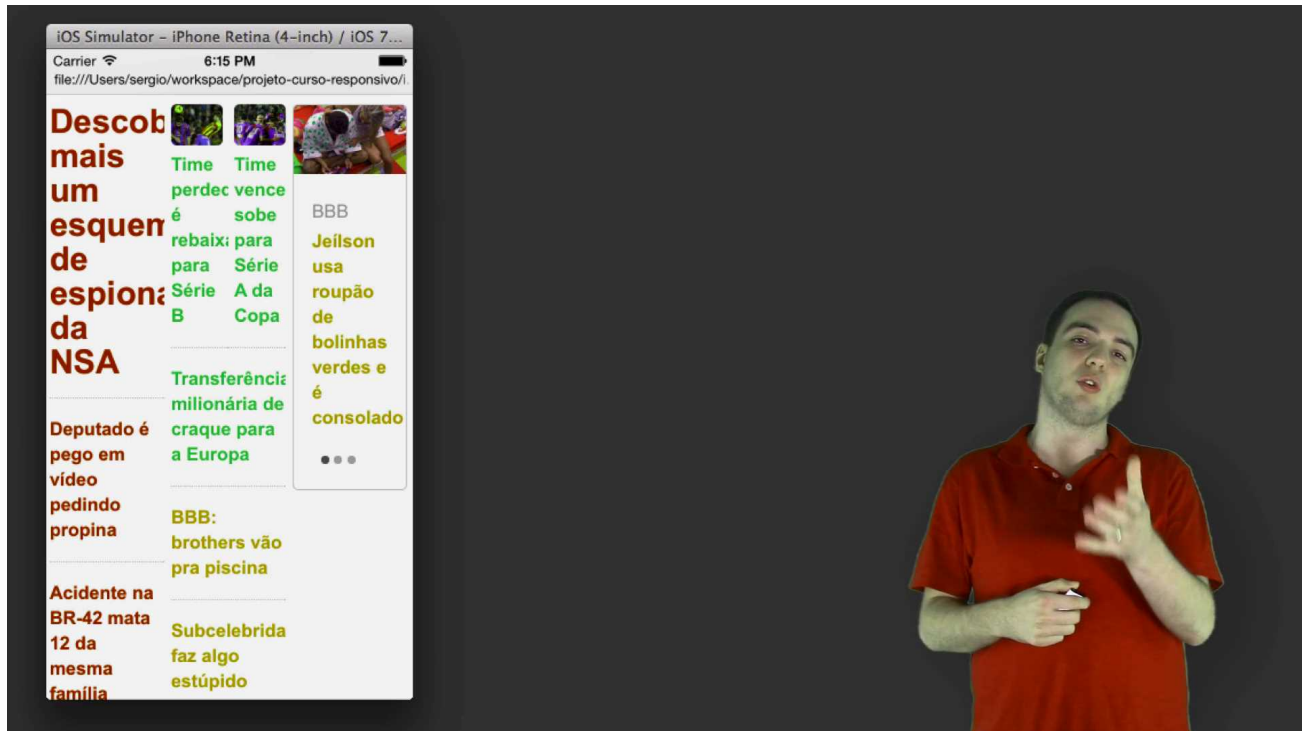


Fig1

Veja-se que o layout está colocado no tamanho de um smartphone. Repare que o limite responsivo é limitado, as medidas flexíveis, as porcentagens, fazendo as três colunas. Contudo, nesse caso as três colunas estão muito apertadas, pois a tela é pequena para os 33% que foram estabelecidos.

Talvez para telas pequenas o interessante fosse parar de trabalhar com três colunas e trabalhar, por exemplo, com uma coluna:



Fig2

Se fosse somente uma coluna, o texto teria espaço para fluir etc. Pode ser que se fizermos com uma coluna, o `scroll` da página vai aumentar, mas isso não é exatamente um problema, pois quando se trabalha com mobile é comum um `scroll` maior, o usuário já está acostumado com isso.

O importante aqui é que, ao mesmo tempo em que o design tem apenas uma coluna no mobile, ele deve ter três colunas na versão desktop, ou seja, queremos dois layouts diferentes dependendo do tipo de dispositivo que se vai trabalhar. Na prática, este é o chamado layout condicional.

LAYOUT CONDICIONAL

Queremos mudar o layout, a partir de alguma condição, que ficará assim quando dito em código: imagine-se uma classe chamada `seção`, que representa cada uma daquelas seções de notícias.

Por exemplo, na versão desktop ele deve ter `width 100%`, ou seja, somente uma coluna.

Em uma versão intermediária, um tablet, por exemplo, ele terá `width 50%`, o que resultará em duas colunas.

Já na versão mobile pequeno, o `width` será `33,33%`, trabalhando com três colunas.

```
.secao {  
  width: 100%;  
}  
  
.secao {  
  width: 50%;  
}  
  
.secao {  
  width: 33.33%;  
}
```



Fig3

Imagine-se esses três códigos, todos corretos, porém cada um para um tipo de cenário específico: desktop, mobile e tablet.

A intenção é aplicar esse código diferente em cada uma das situações, o que pode ser feito por meio das media queries.

```
.secao {  
  width: 100%;  
}  
@media (min-width: 768px) {  
  .secao {  
    width: 50%;  
  }  
}  
@media (min-width: 1024px) {  
  .secao {  
    width: 33.33%;  
  }  
}
```



Fig4

Primeiramente se envolve os blocos de código em blocos de `min-width`.

O que seria o `@media (min-width: 768px)` ? Este bloco está indicando que aquele determinado pedaço de código se aplica somente em telas que tenham, no mínimo, uma largura de 768 pixels ; e no código seguinte, `@media (min-width: 1024px)` , uma largura de, no mínimo, 1024 pixels .

Percebam que se abrirmos o site em um celular pequeno, nem 768px, nem 1024px, valerá somente a primeira regra: `width 100%` .

Se o site for aberto em uma tela um pouco maior, que tenha por volta de 768 px ou 800px, uma média entre as duas media queries, valerá o código intermediário `width 50%` .

Ao se abrir o site em uma tela de desktop, valerá o terceiro código `width 33,33%` , ficando com três colunas.

Esse procedimento é chamado de `CSS3 Media Queries` .

CSS3 MEDIA QUERIES

Trata-se de um recurso do `CSS` que permite implementar a ideia de layout condicional. Um grande pilar do design responsivo. Pois não é possível, na grande maioria dos casos, fazer um design responsivo somente com o design fluído, é preciso também das media queries.

SINTAXE

Quanto à sintaxe das media queries, vejamos que primeiramente deve ser colocado um bloco `@media`, o qual recebe como parâmetro a condição que será testada. Na Fig.5 temos um `min-width: 400px`:

```
@media (min-width: 400px) {  
  
}
```



Fig5

Somente para entender o histórico dessa sintaxe, lembremos que isto foi inspirado no procedimento das `media types` do CSS2 , já existindo há um bom tempo.

CSS2 Media Types

```
@media screen {  
  .ads {  
    display: inline-block;  
  }  
}  
  
@media print {  
  .ads {  
    display: none;  
  }  
}
```




Fig6

A diferença é que naquele sistema não se podia fazer, por exemplo, uma busca por `width` . Deveria ser feito como na Fig. 6, onde somente era possível definir qual o tipo de mídia que está se acessando, como, por exemplo, um `media screen` , uma tela. Então quando for acessado em uma tela, os anúncios são exibidos. Já o `media print` , a mídia de impressão, quando o usuário quer imprimir, os anúncios ficam escondidos, afinal um anúncio em uma impressão talvez não faça muito sentido.

O `@media print` é bastante comum na construção de sites há muito tempo, tendo sua origem nos `media types` . Mais recentemente ocorreu a extensão do `@media screen` , como se vê na Fig. 7, um exemplo em que não as duas coisas não estão misturadas:

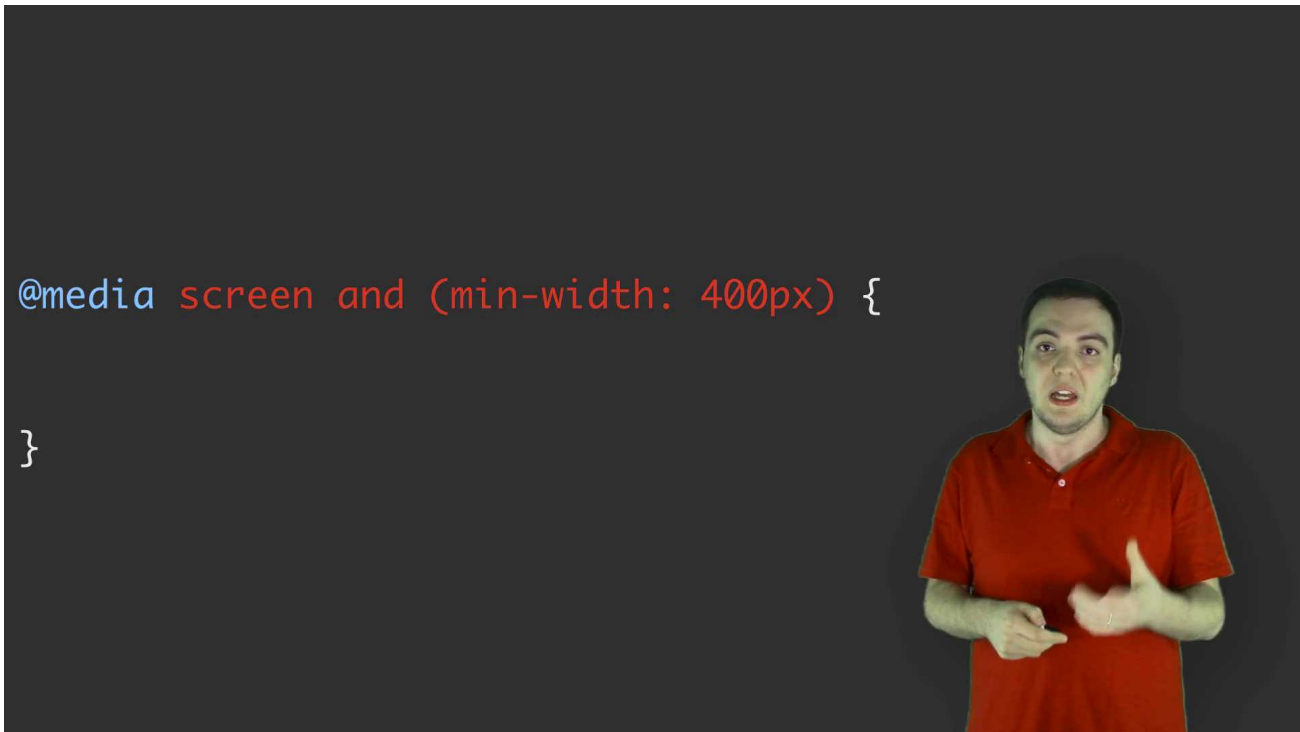


Fig7

De onde se retira, que a `media` `querie` vai avaliar verdadeira se estiver em uma tela, digamos que a impressão não valeria, cujo tamanho seja, no mínimo, 400px e qualquer outro valor acima de 400.

Funciona bem a mistura de `media types` com `media queries`, devendo-se lembrar do cuidado com `browsers` muito antigos, como, por exemplo, um `IE6`, que ainda não suporta as novidades das `media queries`, mas já suportava as `media types`. Devido a essa posição intermediária, um código como aquele apresentado na Fig. 7, pode ser interpretado como somente `@media screen`, ignorando o restante porque não entende e avaliando como verdadeiro [...].

Nas `media queries` novas essa sintaxe foi modificada para alocar a palavra-chave `only`. Na prática nada muda, fazendo apenas a filtragem dos navegadores antigos, ou seja, eles não vão entender o `only`, passando a ignorar o `media query` inteiro, o que estaria mais correto.

Quanto à Fig. 7, lembre-se que ela foi apresentada somente porque é muito comum em vários tutoriais, na internet, encontrarmos essa sintaxe, `only screen and (min-width...)`, embora, durante a construção de um site, não a usemos muito.

Voltando à `media query` inicial (Fig. 5), que tem simplesmente um `min-width`, a indicar somente o tamanho da tela, é suficiente para 99% dos casos. Lembrando que esta `media query` será usada nos códigos e exercícios que faremos.

Temos falado sempre em `min-width`, mas existem várias `media queries` possíveis, como, por exemplo o `max-width`, que é o inverso.

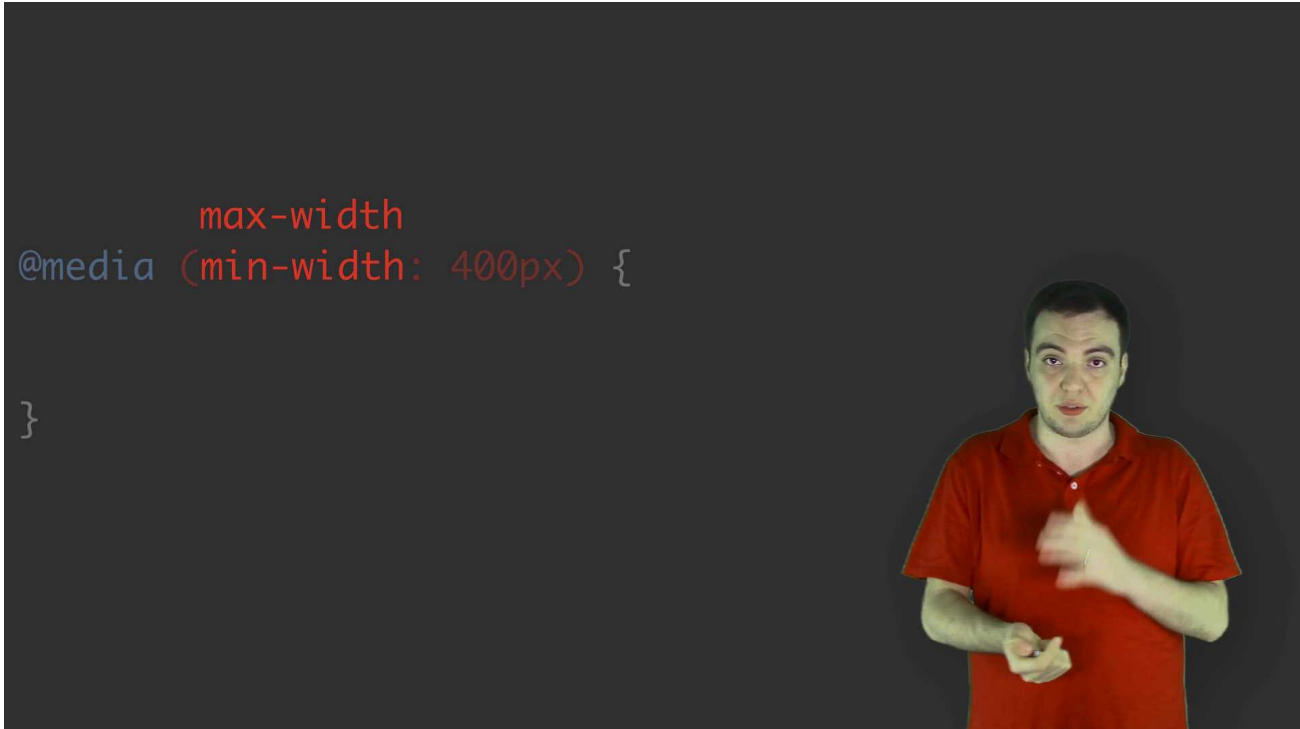


Fig8

Quando se fala em `min-width: 400px`, significa que queremos um tamanho de 400px para mais, ou seja, no mínimo 400px. Por outro lado, quando se coloca `max-width 400px`, indica-se um tamanho máximo de 400px, isto é, de 400px para baixo. Ambas são válidas, dependendo do cenário como veremos mais adiante.

Além do `min` e do `max`, há também o `width` exato, por exemplo, `width 400px`, vai pegar a tela exatamente de 400px.

Existem ainda variações para altura, `height`, e mais duas: `device-width` e `device-height`.

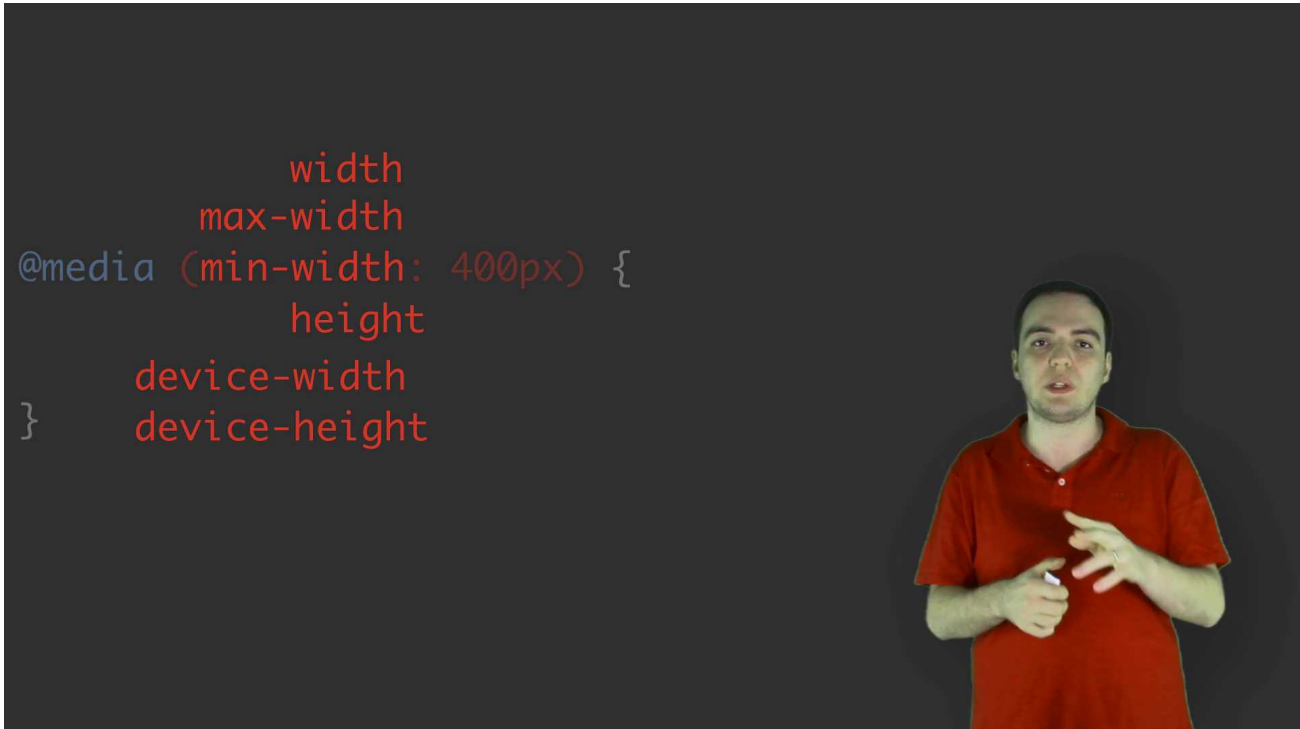


Fig9

A diferença do `device-width` para o `width` é que o primeiro pega a resolução nativa da tela inteira, a tela física que o usuário está trabalhando. Já o `width` pega o tamanho da janela do navegador.

No mobile, geralmente, esses dois valores são os mesmos, mas nem sempre alguns navegadores podem ser redimensionados, notando-se melhor a diferença nas telas de um desktop, por exemplo, em que o `width` ou `height` serão diferentes de seus `device`.

Existem mais algumas media queries que podemos usar, como a orientação retrato ou paisagem.

```
        width
        max-width
@media (min-width: 400px) {
        height
        device-width
        device-height
        orientation: portrait
        orientation: landscape
}
```



Fig10

Podemos pegar um dispositivo que possa ser girado, colocando-o na posição vertical (retrato) ou horizontal (paisagem), utilizando algum recurso a mais no código que está sendo inserido, em um design específico para esses cenários.

A quantidade de media queries existentes é muito grande, mas, na prática, algumas são mais utilizadas.

Como, por exemplo, a `width`. Se tivermos uma `width 400px`, que significa exatamente a largura da tela, exatamente 400px, um pixel a mais ou um a menos, já não funciona. Não parece tão útil, melhor seria retirar a `width` fixa da lista de media queries mais utilizadas. Pois, nesse caso, deseja-se que o CSS se aplique especificamente para uma largura de tela, o que não interessaria no momento em que se estiver trabalhando com uma grande variedade de dispositivos, pois o interessante é ter códigos o mais flexíveis possível. Então retira-se a `width` fixa da lista.

Outra media query que pode ser cortada na maioria dos casos é a `height`. Em geral, na web fazemos conteúdos que a altura é orientada em `scroll`, em que a limitação das páginas se dá pela largura. Não é muito comum uma página com `scroll` horizontal. De uma forma geral, faz-se o `scroll` vertical e a limitação da página é a largura.

As media queries tipo `device` também podem ser colocadas de lado, pois dificilmente haverá interesse pelo usuário do tamanho real da tela do dispositivo utilizado, interessando bem mais o tamanho da tela do navegador, e se este estiver em tela cheia não haverá problemas, os números serão os mesmos. Porém, se o navegador estiver redimensionado, o que interessa no momento de construção da página é saber o tamanho do navegador dos usuários. O tamanho real da tela não seria tão útil, nesses casos.

Já as media queries de orientação são sempre interessantes, devendo ser mantidas.

```
        width
        max-width
@media (min-width: 400px) {
        height
        device-width
        device-height
        orientation: portrait
        orientation: landscape
        ...
}
```



Fig11

Portanto, apesar de existirem várias media queries, nem todas são úteis para nosso propósito.

Especificamente, o `min-width` ainda é mais útil, na prática, que o `max-width`. Embora a utilização de ambos possa ser implementada sem problemas. Voltaremos a esse tópico mais adiante.

Retornando ao `min-width 400px`, como esse valor foi obtido? Qual a origem desses 400px? Ou, ainda, 768px e 1024px (Fig. 4)?

Esses números são denominados `breakpoints`, ou seja, os pontos de quebra do design, indicando o momento em que este será alterado pela media query.

Como exemplo, temos alguns `frameworks` famosos, que foram inspiração para demonstrar esses valores.

FRAMEWORKS RESPONSIVOS

TWITTER BOOTSTRAP

Label	Layout width
Large display	1200px and up
Default	980px and up
Portrait tablets	768px and above
Phones to tablets	767px and below
Phones	480px and below



Fig12

Na Fig. 12 temos o twitter bootstrap , em que existe uma tabela de media queries mais comuns (1200px, 980px, 768px, 480px...).

Mas como se chega nesses valores? Por exemplo, o valor 768px é a largura de um iPad, 480px é a largura de um iPhone clássico, não o mais novo. Já 1200px costuma ser o tamanho de um notebook atual ou de uma tela desktop padrão que possui 1280px, menos as barras de rolagem, resultando em 1200px.

Assim, se atentarmos bem, tais valores são inspirados em dispositivos comuns existentes no mercado.

A seguir, temos o exemplo de outro `framework` :

FRAMEWORKS RESPONSIVOS

320 and up

320px

480px

600px

768px

992px

1382px



Fig13

Onde se vê um valor de 320px and up e mais algumas media queries, inclusive algumas parecidas com a Fig. 12 (320px, 480px, 600px, 768px, 992px e 1382px).

Observe-se que, novamente, temos valores inspirados em dispositivos comuns. Por exemplo, 320px é o tamanho de um iPhone, 480px um iPhone em modo paisagem, 600px é um tamanho comum a Tablets de 7 polegadas, 768px é o tamanho de um iPad, 992px costuma ser os 1024px sem as barras de rolagem e 1382px é do desktop maior. Assim, percebe-se que são utilizados valores de dispositivos comuns.

Na prática, por exemplo, temos que enquanto um iPhone tem 320px, os Androids costumam ter 360px, e não temos uma media query que fique entre esses valores. Aliás, pergunta-se, precisaria de um valor que fosse exatamente a média desses dois valores citados? Ou ainda, quais os recursos que podemos utilizar para gerar os `breakpoints` ?

Aqui devemos lembrar que as tabelas de dispositivos-padrão não são o ideal para a criação de `breakpoints` , mas sim os chamados `breakpoints` de conteúdo.

BREAKPOINTS DE CONTEÚDO

Isso significa dizer que, a página criada deve ser aberta em diferentes dispositivos e verificar em quais deles o tamanho não está ideal para aquela resolução, anotando-se o tamanho e criando uma media query para o dispositivo com uma resolução específica.

Desta forma, verifique como o design se comporta em variadas resoluções, e crie `breakpoints` que ajustarão o design pretendido àquelas resoluções diferentes.

Por isso, não é interessante pegar dispositivos "famosos" ou padrão, pois, na verdade, não existem dispositivos-padrão. Os usuários se utilizam de milhares de dispositivos diferentes e isso muda o tempo todo. Assim, a única maneira de se

garantir o bom funcionamento do site em todas as resoluções é testando em todas elas e ajustar quando for necessário.

A seguir, temos um exemplo de uma página em tamanho pequeno, que será redimensionada para vários tamanhos possíveis.

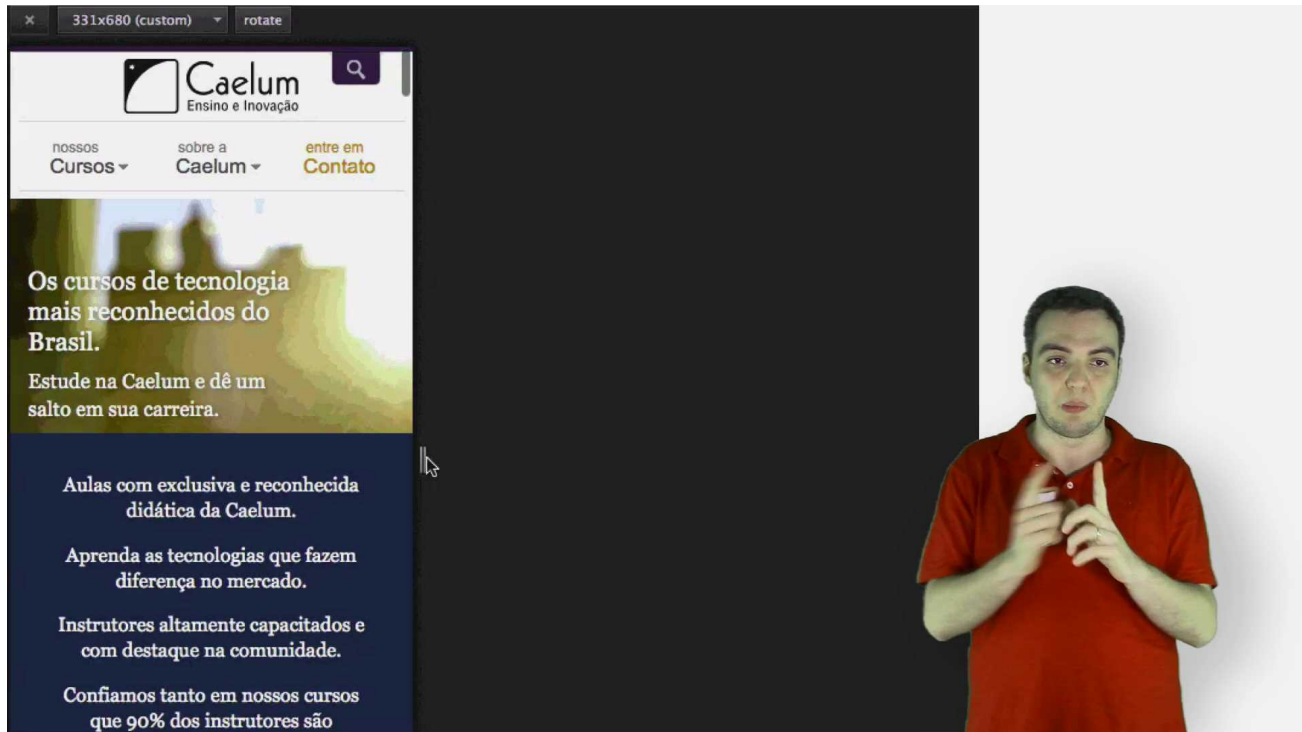


Fig14

No alto pode-se ver qual o tamanho possível da janela e a adaptação do design.

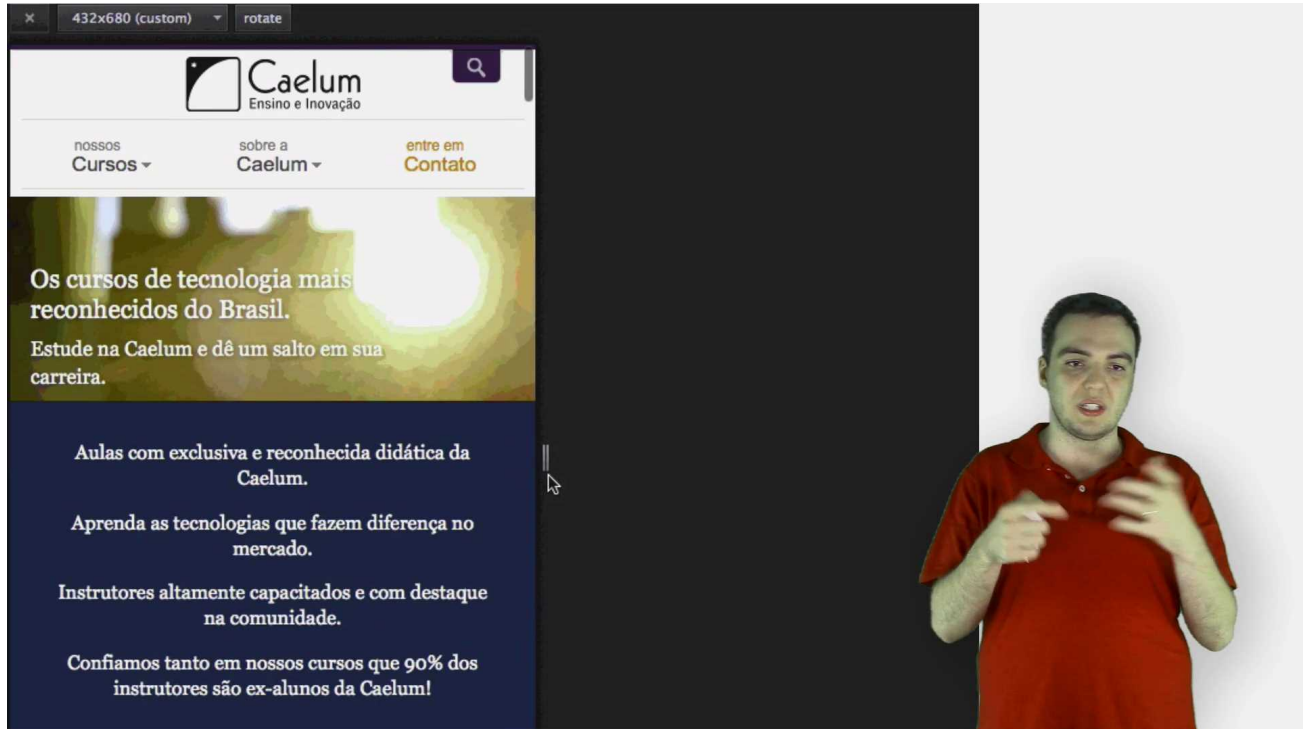


Fig15

Começando com trezentos e poucos pixels percebe-se o design se adaptando.

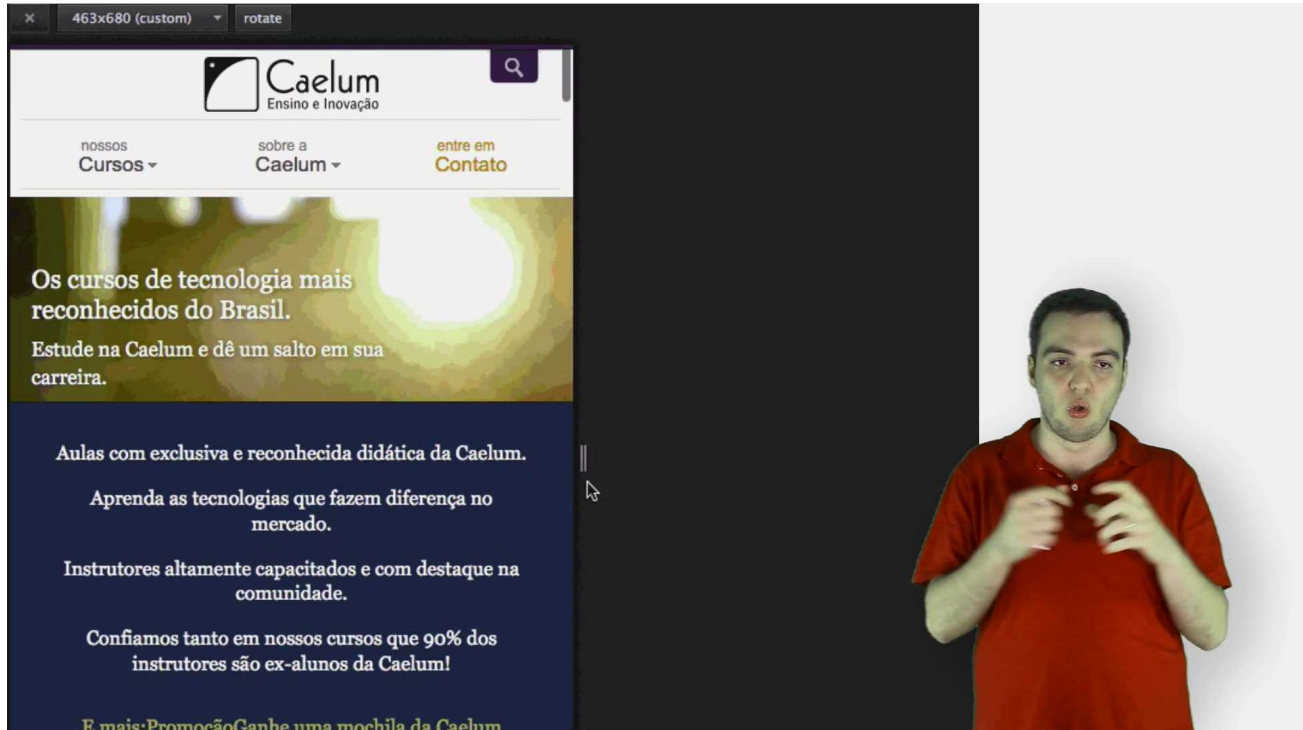


Fig16

O menu, por exemplo, que está embaixo do logo, quando se chega em, mais ou menos 460px, sobe, se posicionando ao lado do logo.



Fig17

Mas 460px não é um número de dispositivos-padrão, sendo utilizado porque o layout fica interessante.

Na Fig.18 pode-se ver como o layout se comporta com 580px.

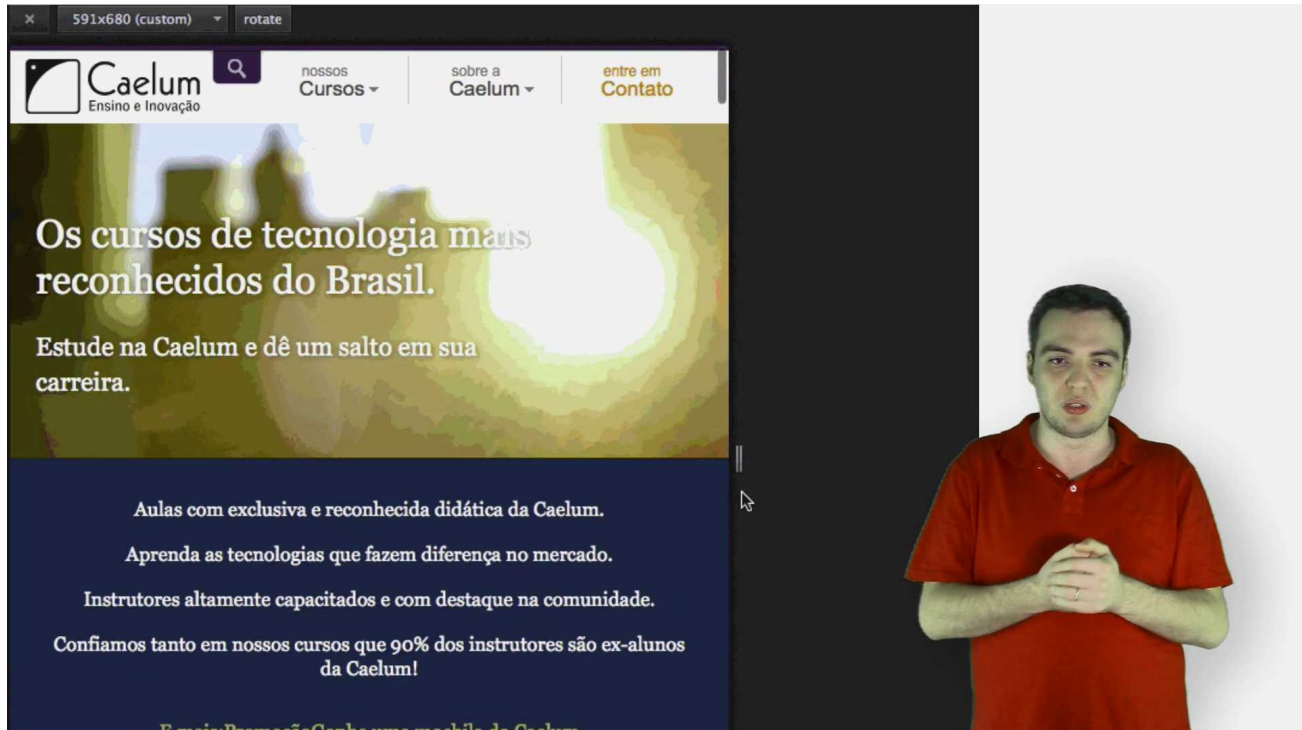


Fig18

E ao aumentar para algo mais que 600px, o design adquire outra configuração, deixando o conteúdo maior.



Fig19

Aos 670px, já se ajusta o tamanho de texto e assim por diante.



Fig20

Portanto, a ideia é aumentar o tamanho da página e percebendo as suas necessidades, de modo que encontro o breakpoint ideal para cada resolução.

Esse é o procedimento da criação de media queries para breakpoints de conteúdo. E para realizar esses testes alguns navegadores já possuem a possibilidade de simular os dispositivos móveis colocando qualquer resolução desejada como no [chrome](https://developers.google.com/web/tools/chrome-devtools/device-mode/?hl=pt-br) (<https://developers.google.com/web/tools/chrome-devtools/device-mode/?hl=pt-br>) e no [firefox](https://developer.mozilla.org/pt-BR/docs/Tools/Modo_Design_Adaptavel) (https://developer.mozilla.org/pt-BR/docs/Tools/Modo_Design_Adaptavel).

Como último detalhe para fechar esse assunto de media queries, trataremos daquela diferença entre o `max-width` e o `min-width`, que é fazer media queries no estilo `mobile first`.

MEDIA QUERIES MOBILE FIRST

Utilizando a media query para criar um determinado breakpoint em uma certa categoria de dispositivos. É possível escolher se dentro da media query vai um mobile ou um desktop, ou seja, a versão padrão do CSS será mobile ou desktop?

Agora veremos como funciona o estilo `mobile first`. Observemos a próxima figura:

MQ desktop-first

```
h1, h2 {  
  float: left;  
}  
  
@media (max-width: 400px) {  
  h1, h2 {  
    float: none;  
  }  
}
```



Fig21

Foi criado um media query chamado `desktop first`, em que existem dois títulos, `h1` e `h2`, ambos devem ficar lado a lado na tela do desktop, flutuando, e no mobile um deve escorregar para baixo do outro, sem flutuar. Ou seja, queremos `float left` em um cenário e `float none` em outro.

No formato `media query desktop first`, o código fora do media query, com `float left`, representa o que se quer para o desktop: os dois elementos flutuando.

Em seguida, criaremos uma media query de `max-width` para representar o que se quer na versão mobile, isto é, para dispositivos de até 400px que terão o `float none`. Esta é a versão `desktop first`.

Se invertermos a ordem das media queries, ou seja, se o código padrão passa a ser o mobile e o código interno da media query passa a ser o desktop, será necessária a inversão da media query de `max` para `min-width`.

Porém, há um detalhe interessante:

MQ desktop-first

```
h1, h2 {  
  float: left;  
}
```

```
/* float:none; */
```

```
@media (max-width: 400px) {  
  h1, h2 {  
    float: none;  
  }  
}
```

MQ mobile-first

```
@media (min-width: 400px) {  
  h1, h2 {  
    float: left;  
  }  
}
```

Fig22

Repare que, como o padrão do `float` já é trabalhar com `float none`, não é necessário escrever `float none` na versão mobile. Portanto, escreveremos somente `float left` na versão desktop, ou seja, de 400px para cima.

Assim, ao usarmos o `min-width` é muito comum a aplicação em cenários como esse, em que o padrão já vale para o `mobile`, enquanto são criadas para o `desktop` customizações a mais, afinal é mais comum que o layout do `desktop` seja mais complicado que o do `mobile`, em decorrência de mais espaço, de uma largura maior.

Na prática, a media query de `min-width` é muito mais utilizada que um `max-width`, pois o `min-width` representa a estratégia da `media queries mobile first`.

RESUMO

- 1) Foi visto que as media queries são uma forma de lidar com layout condicional, ou seja, layouts diferentes, ajustes diferentes, dependendo de alguma condição, em especial tamanho de tela;
- 2) Também foi visto como eram as `media types`, no `CCS2`, e como é a sintaxe atual das media queries no `CCS3`;
- 3) Além disso, foram discutidas as boas práticas, como trabalhar com `breakpoints`, a partir do conteúdo da página que se deseja criar, do layout desta página, do seu design, o que é melhor do que recorrer a tabelas de dispositivos considerados "padrão";
- 4) Por fim discorremos sobre a `media query mobile first`, em que é utilizado mais o `min-width`, é mais interessante na prática.

