



Artigo

Dominando Design Responsivo com HTML5 e CSS3

Esse artigo explora táticas que os desenvolvedores web podem usar para criar páginas responsivas, usando apenas HTML5 e CSS3 sem ter de recorrer a frameworks de componentes.

[Marcar como concluído](#)[Anotar](#)[Artigos](#)[HTML e CSS](#)[Dominando Design Responsivo com HTML5 e CSS3](#)

Desenvolver aplicações web hoje em dia muitas vezes se resume à correta escolha do framework de componentes (como Bootstrap, Google Material Design Lite, etc.) para facilitar a construção da estrutura e estilo das páginas. Esse cenário funciona muito bem até o momento em que tais bibliotecas não atendem mais aos recursos específicos do seu projeto: por exemplo, quando precisamos de componentes/design muito específicos ou, pior ainda, quando precisamos misturar os componentes de duas bibliotecas diferentes numa só aplicação para atender a demanda, aumentando consideravelmente a quantidade





todo o website. Este artigo vai de encontro à essa realidade, ensinando a fazer uso dos recursos mais recentes da HTML5 e CSS3 para construir seus frameworks web.



Guia do artigo:

- [O poder do Sass](#)
- [Configurando o Sass](#)
- [Sass vs SCSS](#)
- [HTML5](#)
- [Meta tags](#)
- [Exemplo prático](#)
- [Grids CSS](#)
- [Tipografia](#)

Sem dúvida a **responsividade** é o conceito que define a qualidade de um site, aliada a conceitos como confiabilidade, segurança, performance e preocupação dos criadores para com seu público. Não dá mais para pensar em criar qualquer aplicação web ou website sem priorizar a responsividade como pré-requisito básico e fundamental.

O próprio Google, bem como outras ferramentas de busca na web, ranqueiam melhor sites que apresentam seu conteúdo de forma *mobile friendly*, isto é, preparam seus conteúdos para serem exibidos em dispositivos de diversos tamanhos, a incluir smartphones, tablets, notebooks, etc.

Em ferramentas e metodologias de SEO (*Search Engine Optimization*) esse conceito é tido como primário para um bom resultado junto aos sites de pesquisa, logo, você deve estar preparado para lidar com isso em seus projetos front-end como um todo.

Frameworks famosos como o Twitter Bootstrap, Angular Material (do AngularJS) e o novo Google Material Design Lite já trazem toda uma especificação de componentes HTML.





([jQuery](#), [AngularJS](#), etc.) e HTML puro, via build automatizado, usando o Grunt, por exemplo.



o design da aplicação como um todo seja completamente diferente do fornecido por esses frameworks, ou casos em que os componentes das mesmas não sejam suficientes para desenvolver as peculiaridades que seu projeto exige, dentre outros.

Nestes casos, o designer, ou desenvolvedor front-end deve ter embasamento o suficiente para lidar com esse tipo de desenvolvimento, de repente criando seu próprio framework de componentes, classes e funções. E são exatamente estes pontos que esse artigo visa tratar, explorando conceitos como grids, [desenvolvimento com Sass](#), tipografia, dentre outros.

O poder do Sass

Para fins de simplicidade e produtividade, usaremos o Sass para facilitar a implementação do CSS nos exemplos do artigo.

O Sass permite que elementos sejam aninhados para flexibilizar a forma como as regras serão geradas no CSS final. Por exemplo, vejamos o exemplo demonstrado pela **Listagem 1**.

A primeira parte dela mostra um [código CSS](#) feito em Sass que define a forma de exibição de um elemento de classe “barra-navegacao” via `display` com valor `flex` (que diz que o conteúdo deve se adaptar às dimensões da tela onde for exibido), bem como o `padding` dos elementos de lista `li` que estiverem inseridos dentro do anterior.

Na segunda parte, vemos o resultado da compilação desse Sass, ou seja, o CSS final gerado. Veja como o Sass simplifica a criação de regras de estilo ao evitar que tenhamos de duplicar código sempre que quisermos uma regra em herança.





```
3     display: flex;
4     li {
5         padding: 5px 10px;
6
7
8 // Depois
9 .barra-navegacao {
10     display: flex;
11 }
12 .barra-navegacao li {
13     padding: 5px 10px;
14 }
15 }
```



O Sass se baseia num conjunto de vários conceitos para implementar CSS, vejamos:

- Ele pode ser baseado em duas tecnologias diferentes: Ruby ou LibSass (**BOX 1**). Vamos usar neste artigo o Ruby por questões de simplificação.
- Ele é um gem no Ruby, isto é, um pacote usado no Ruby.
- Podemos executá-lo via interface de linha de comando, mas também é possível sua execução via aplicações de terceiros, com a devida importação de suas bibliotecas dependentes.
- Ele é uma linguagem de scripting convencional, como JavaScript, CoffeeScript, etc.
- O Sass também contribui para a eliminação das repetições de código quando desenvolvemos CSS puro. Isso se dá por intermédio da sua sintaxe hierárquica que permite o reaproveitamento de regras para os elementos que pertencerem à mesma. Veremos mais detalhes sobre isso na prática adiante.
- Parte do fluxo de trabalho do Sass é “assistir” a um arquivo de tipo SCSS, por exemplo, `estilos.scss`. Quando ele detecta uma mudança nesse arquivo, um novo arquivo `estilos.css` é automaticamente gerado e compilado.



Configurando o Sass



Precisamos de basicamente três passos para ter o Sass instalado:

1. Baixar e executar o instalador do Ruby.
2. Abrir um terminal de comandos referente ao seu SO.
3. Instalar a gem do Sass.

Para verificar se o Ruby já está instalado na sua máquina, digite o seguinte comando no cmd:

```
1 | ruby -v
```

Uma mensagem com a versão do mesmo, bem como a data da *revision* e versão do instalador para o seu SO (32 ou 64 bits) será impressa no console:

```
1 | ruby 2.1.5p273 (2014-11-13 revision 48405) [x64-mingw32]
```

Caso o leitor já esteja com uma versão antiga do Ruby instalada é aconselhado remove-la antes de instalar a nova, a fim de evitar problemas de conflito no ambiente. Para o Windows, especificamente, precisamos do *RubyInstaller for Windows* (vide seção **Links**).

Quando finalizar o download, execute o arquivo, selecione a língua para o passo a passo da instalação, aceite os termos de licença e, na próxima tela, selecione o diretório onde deseja



No final, antes de executar qualquer comando com o novo Ruby precisamos fechar a janela do terminal e abrir uma nova. Então execute mais uma vez o comando de verificação da



```
1 | ruby 2.2.3p173 (2015-08-18 revision 51636) [x64-mingw32]
```

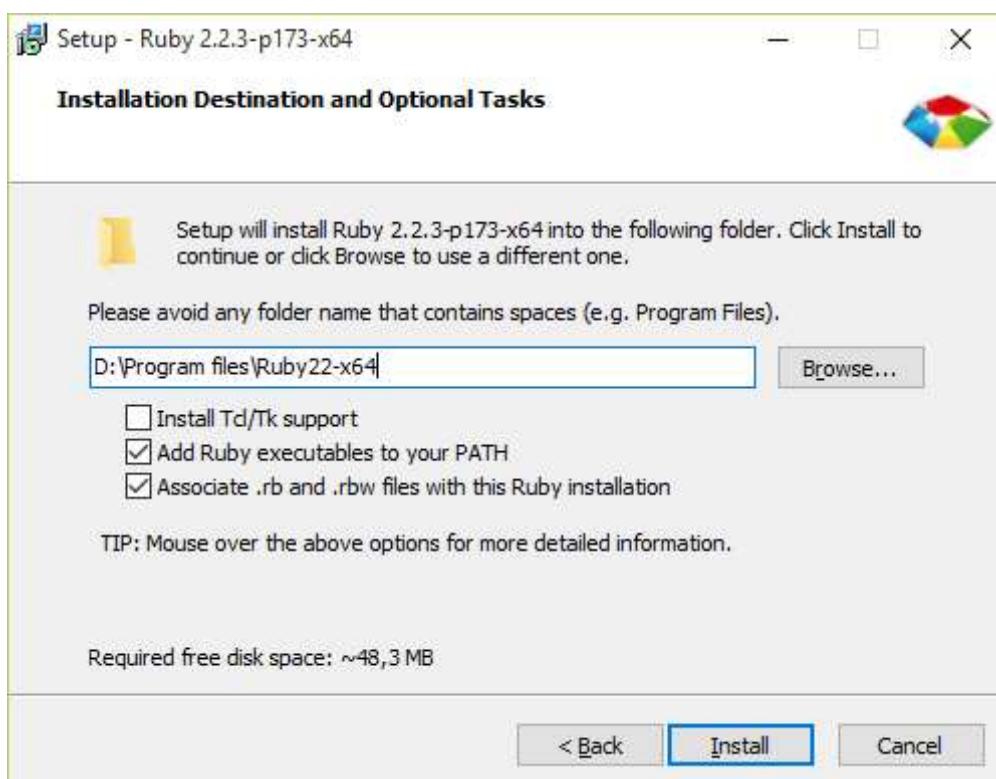


Figura 1. Tela de configuração das opções do Ruby.

Com o Ruby ok, agora é hora de instalar a gem do Sass no ambiente. Para isso, execute o seguinte comando no terminal:

```
1 | gem install sass
```





Para trabalhar com o Sass, precisamos ter um mínimo de organização de diretórios possível. Portanto, selecione um diretório de sua preferência para o projeto e crie a estrutura de pastas demonstrada na **Listagem 2**. Veja que temos duas pastas de estilo em detrimento dos arquivos de Sass não compilados, e CSS pós compilados.

Listagem 2. Estrutura de diretórios do projeto.

```
1 | +-responsive-web
2 |   +---- html
3 |   +---- img
4 |   +---- js
5 |   +---- style
6 |     +--- CSS
7 |     +--- SCSS
```

Navegue até a pasta `/scss` e crie um novo arquivo de nome `estilos.scss`. No terminal cmd, usando o comando `cd`, navegue até a pasta `/style` e execute a seguinte instrução:

```
1 | sass --watch scss:css
```

Esse comando é responsável por dizer ao Sass que ele deve “assistir” as pastas informadas à frente (scss/css), checando quais arquivos do Sass existem, e gerando automaticamente o





Sass vs SCSS



Existem duas sintaxes padrão para escrever código de estilo usando o Sass: a **Sass** e a **SCSS**.

A primeira foi, por muito tempo, a única forma disponível para tal finalidade, porém era considerada muito diferente do CSS original, aumentando assim a curva de aprendizado na tecnologia, mesmo para quem já tivesse familiaridade com o mesmo.

Vejamos um exemplo bem simples na **Listagem 3** que define uma propriedade de `float` para um seletor **a**, e uma propriedade de cor (`color`) da fonte para o mesmo seletor e para um outro seletor **b**.

A sintaxe Sass se baseia em “espaços em branco” para definir a hierarquia dos elementos do CSS final. Quando criamos o seletor-b com um espaçamento maior que o do seletor-a dizemos ao Sass que a regra se aplica a ambos.

Caso contrário, se o seletor-b estivesse no mesmo nível do seletor-a, teríamos a regra aplicada apenas ao primeiro. O CSS de resultado pode ser visto na mesma listagem.

Veja como o procedimento não é tão intuitivo quanto usar a sintaxe do SCSS, como podemos ver na **Listagem 4**. Nele temos o uso do novo símbolo `&`, que nos permite adicionar o nome do seletor pai aos seletores aninhados sem ter de digitar o nome completo.

Em outras palavras, basta mapear o prefixo do seletor e, dentro do mesmo, concatenar cada um dos nomes via operador `&`; assim, o SCSS fará todo o trabalho de montagem que acabará findando no mesmo resultado do exemplo com Sass.

Listagem 3. Exemplo de CSS gerado a partir de um Sass.

1 | // Antes





```
7 // Depois
8 .seletor-a {
9
10
11
12 .seletor-a, .seletor-b {
13   color: red;
14 }
```



Listagem 4. Exemplo de CSS gerado a partir de um SCSS.

```
1 // Antes
2 .seletor- {
3   &a {
4     float: left;
5   }
6   &a, &b {
7     color: red;
8   }
9 }
10
11 // Depois
12 .seletor-a {
13   float: left;
14 }
15 .seletor-a, .seletor-b {
16   color: red;
17 }
```

HTML5

A HTML é uma parte essencial de ser assimilada e entendida quando do **desenvolvimento de código responsivo**. Diante disso, muitas tags padrão da HTML5 que existem para nos auxiliar são simplesmente ignoradas pelos desenvolvedores e designers.



	  		<ul style="list-style-type: none">■ O conteúdo principal da página deve ser incluído nessa tag.■ Deve ser exclusivo e único.■ Essa tag não deve ser inserida nas tags: <code><header></code> , <code><footer></code> , <code><nav></code> , <code><aside></code> ou <code><article></code> .■ Deve haver apenas uma dela por página.
<code><main></code>	Pode ser usada como um container para os conteúdos principais do documento. Geralmente está relacionada à marcação de um tópico central de uma seção/funcionalidade da aplicação. Esse conteúdo deve ser único no documento.		<ul style="list-style-type: none">■ Um <code><article></code> pode ser aninhado em outros <code><article></code> .■ Pode existir mais de um por página.
<code><article></code>	Representa uma composição autocontida num documento, página, aplicação, ou site, que deve ser distribuível ou reusável de forma independente. Pode ser um post de fórum, um artigo de revista ou jornal, uma entrada num blog, ou qualquer outro item independente de conteúdo.		



Cabeçalho (`<h1>` - `<h6>`) como elemento filho.

de um elemento



`<article>`. Além disso, é bom



um cabeçalho `<h1>` - `<h6>` em cada uma.

- Pode existir mai

`<aside>`

Representa uma seção com conteúdo conectado lateralmente ao corpo da página. O mesmo pode ser considerado separado do resto do conteúdo. Geralmente são representadas como `sidebars` ou `inserts`. Bons exemplos de uso dela incluem: biografia de um autor, informações de perfil de usuário, ou links relacionados num blog.

- Se o conteúdo não se encaixar na tag `<main>`, ele certamente poderá ser inserido num `aside`.
- Pode existir mais de um por página.

`<header>`

É comum pensar que a seção de topo da nossa página é o `header` (cabeçalho), e que isso é correto. Entretanto, o nome correto para essa parte do site é `masthead`, ou seja, o `header` principal da página, que geralmente contém a logo, alguma navegação, talvez um campo de pesquisa, etc. Já o `header` pode ser considerado o cabeçalho de qualquer seção, portanto, podemos ter vários deles.

- Uma boa regra é sempre usar o `<header>` dentro de uma `<section>`.
- Você pode mapear um `heading` (`<h1>` - `<h6>`) em um `header`, mas não é comum, tampouco necessário.



pagina.

■ Deve sempre



informações sobre o elemento pai que o contém.

- Apesar do nome “footer” se referir à parte de baixo de uma página, artigo ou aplicação, esse elemento não necessariamente tem que estar nesta parte.
- Pode ter mais de um por página.



<footer>

Representa o rodapé para a sua seção mais próxima, ou secciona o elemento raiz da página. Tipicamente contém informações sobre autor da seção, dados de *copyright*, ou links para documentos relacionados.

<nav>

- Representa uma seção da página que conecta a outras páginas ou a outras partes da mesma página: uma seção com links de navegação, por exemplo.
- Deve ser usada para agrupar uma lista ou coleção de links, externos ou internos.
- É uma prática comum usar listas não-ordenadas (``) dentro desse elemento





mais simples

para o design.

- Também é



essa tag em um elemento

<header>, mas não é requerido.

- Nem todos os links são requeridos de estar dentro desse elemento. Por exemplo, se tivermos links no rodapé, não é preciso inserir uma <nav> dentro do <footer> .
- Pode existir mais de um por página.



Tabela 1. Lista de tags usadas para incutir responsividade.

Outro conceito importante para a definição de uma correta responsividade nas páginas HTML é o de `roles` do WAI-ARIA (*Web Accessibility Initiative – Accessible Rich Internet Applications*), um padrão internacional que visa implementar práticas de acessibilidade nas aplicações ricas da web.

As suas `roles` (ou regras), também chamadas de *ARIA roles*, servem para definir



1 | <header role='banner'>



Existem muitos tipos diferentes de `roles` que podem ser implementadas, mas vamos focar apenas nas mais importantes para o conceito acessibilidade responsiva. A **Tabela 2** ilustra esta relação, com suas respectivas características.

Role	Descrição
banner	<ul style="list-style-type: none">■ É usualmente aplicada à tag <code><header></code> do topo da página.■ Geralmente, o conteúdo que tem um <code>role="banner"</code> aparece constantemente ao longo do site, em vez de apenas em uma página específica.■ Apenas uma é permitida por página.
navigation	<ul style="list-style-type: none">■ É usualmente aplicada ao elemento <code><nav></code>, mas pode também ser aplicada a outros containers como <code><div></code> ou <code></code>.■ Descreve um grupo de elementos/links navegáveis (internos ou externos).■ Pode ter mais de um na página.
main	<ul style="list-style-type: none">■ É usualmente aplicada ao elemento <code><main></code> da página.■ O container principal/central da página deve ser marcado com essa <code>role</code>.■ Apenas uma é permitida por página.
contentinfo	<ul style="list-style-type: none">■ É usualmente aplicada ao elemento <code><footer></code> da página.■ É a seção que contém informações sobre o documento/site/app.■ Apenas uma é permitida por página.





Se o elemento `<form>` estiver mapeado dentro de uma `<div>`, essa `role` também pode ser aplicada à mesma. Se for o caso, não há necessidade de adicionar a `role` ao `form` novamente.



form

- É usualmente aplicada ao elemento `<div>` que contém algum tipo de formulário, exceto o de `search` que acabamos de ver.
- Não deve ser aplicada ao elemento de `<form>` atual, porque o mesmo já tem uma semântica de `role` padrão que suporta essa tecnologia.

complementary

- É usualmente aplicada ao elemento `<aside>`.
- Deve ser usada em uma região que contém suporte a conteúdo.
- Pode haver mais de uma por página.

Tabela 2. Lista de roles usadas pela WAI-ARIA.

Meta tags

As *meta tags* são estruturas importantes, principalmente no SEO, para marcar coisas como autoria das páginas, versionamento, se a página é responsiva ou não, bem como enviar certos comandos aos navegadores que identificam o conteúdo e melhoram a exibição do HTML. Veja na **Tabela 3** uma relação das tags (e suas descrições) mais importantes para **implementar design responsivo** nas páginas web.

Meta Tag	Exemplo	Descrição
viewport	<code><meta name="viewport" content="width=device-</code>	<ul style="list-style-type: none"> ■ É a meta tag mais importante para o design responsivo.



- A propriedade `width` define o tamanho do `viewport`. O valor “`device-width`” diz ao browser para ocupar todo o espaço da tela



um valor em pixels.

- A propriedade “`initial-scale`” define o nível de zoom sob o qual a página deve ser exibida. 1 é igual a 100% de zoom e 1.5 igual a 150%, por exemplo.

X-UA-Compatible

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

- Visa estabelecer compatibilidade com o Internet Explorer, por isso é aplicada apenas a esse navegador.
- A diretiva “`http-equiv`” diz ao IE que uma certa engine de renderização precisa ser usada para carregar a página.
- A diretiva “`contente`” diz ao IE que ele deve usar suas engines de HTML e JavaScript mais recentes.

charset

```
<meta charset="utf-8">
```

- Essa é talvez a mais conhecida pelos desenvolvedores, já que se não inserida na página, os caracteres especiais e acentos gráficos aparecerão distorcidos.
- Ela diz ao browser que conjunto de caracteres deve ser usado para interpretar o conteúdo.
- Outro valor muito comum é “`ISO-8859-1`”, mas o `UTF-8` é mais aconselhado pois há mais chances de o seu browser o interpretar.





Para exemplificar os conceitos até aqui expostos, vamos implementar uma página inteiramente responsiva, com a inclusão de código HTML5 organizado com base nas tags



página em formato de site convencional, com um cabeçalho (e barra de navegação, menus, caixa de pesquisa), conteúdo principal (com um artigo, cabeçalho interno, formulário de contato) e um rodapé (com *copyright* e links de rodapé).

Comecemos então pelo cabeçalho da página. Crie um novo arquivo HTML de nome index.html no diretório raiz do projeto e insira ao mesmo o código da **Listagem 5.**

Ela traz apenas uma implementação simples da tag `<head>` da página HTML, com as respectivas *meta tags* que citamos e foram antes inseridas, um título (tag `<title>`) e o arquivo de estilo CSS que criamos sendo importado.

Ainda precisamos criar o conteúdo deste último. Observe também que, em vez das convencionais tags `<div>` usadas para dividir as seções das páginas, estamos usando a tag `<header>` dentro do corpo HTML para demarcar o cabeçalho único e principal do site. Demos a ele uma classe CSS de nome “masthead” e a `role` de “banner”, já que será único também.

Dentro do `header` temos duas `div`: a primeira contém o título do cabeçalho propriamente dito e a segunda guarda o formulário de pesquisa que, por sua vez, contém a `role` de “search” (única por página). Para todos os campos de texto da página daremos a classe CSS “field” para padronizar no Sass.

Listagem 5. Cabeçalho HTML da nossa página do site.

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
```





```
10 | </head>
11 |
12 | <body>
13 |     
14 |   ...
15 |   <div class="search" role="search">
16 |     <form>
17 |       <label>Busca:
18 |         <input type="text" class="field">
19 |         <button>Buscar Agora!</button>
20 |       </label>
21 |     </form>
22 |   </div>
23 | </header>
24 |
25 |   <!-- Restante do conteúdo -->
26 |
27 | </body>
28 |
29 | </html>
```

Com o HTML pronto precisamos agora criar o conteúdo Sass para lidar com o design inicial da página. Comecemos então pelo Sass de propriedades globais, tais como cor do plano de fundo, logo, cabeçalho e alguns atributos globais que serão comuns a todos os elementos da página. Insira o código da **Listagem 6** no arquivo `estilos.scss`.

Veja que no início dela criamos algumas declarações globais. No Sass é possível declarar variáveis, tal como fazemos nas linguagens de programação como JavaScript, por exemplo, e usar seus valores em todo o restante do código. Isso facilita a padronização uma vez que não precisamos repetir a cor da fonte, por exemplo, para toda nova regra CSS criada.

As cores abrangem o plano de fundo, fontes, cabeçalhos, etc. Em seguida, definimos uma função `mixin` (vide **BOX 2**) do Sass de nome `paraTelasPequenas()`, a qual será responsável por traduzir as regras criadas mais abaixo para dispositivos pequenos, por isso a divisão `por 16px em no atributo @media`. Também se crie este atributo é reconhecível por delimitar o





- **Regras globais:** definem propriedades comuns do CSS para fonte do texto, cor de fundo (via variável do Sass `&ba`), cor dos cabeçalhos `<h1>` a `<h6>` e estilo dos ícones    específicos para telas de até 420px).
- **Header:** definimos o estilo do cabeçalho, que deve estar alinhado ao centro com seus elementos também centralizados, uma logo (somente texto) e cor de fundo (note que estamos usando o mesmo estilo do portal da DevMedia).
- **Placeholder:** essa é a primeira herança no Sass que usamos. Esse elemento contém regras CSS universais que todos os elementos de seção também terão. Dessa forma, eles só precisam estender do `placeholder` e as mesmas regras serão aplicadas.
- **Campo de busca:** logo na classe “search” já herdamos do `placeholder` que criamos. O restante das configurações é intuitivo.
- **Formulários:** regras básicas para que o formulário ocupe 100% do espaço disponível.
- **Elementos de formulário:** configura o estilo para os campos de input, textarea e botões (repare que para este último definimos seu estilo separadamente, e não dentro de cada regra que tenha um botão qualquer, deixando o mesmo universal).

Como o “watcher” do Sass já foi ativado, basta salvar a página HTML e abri-la em um navegador web. O resultado da página em um browser desktop pode ser visto na **Figura 2**, enquanto a visualização em dispositivos menores se encontra na **Figura 3**.

Listagem 6. Código de SCSS global do Sass.

```
1 ///////////////////////////////////////////////////////////////////
2 // Declarações Customizadas
3 // Cores
4 $bk: #272822;
5 $r: #c03;
6 $g: #429032;
7 $b: #8cc53e;
8 $y: #49c5bf;
```





```
13 }  
14  
15 //Globais  
16  
17     
18 *:after {  
19   box-sizing: border-box;  
20 }  
21  
22 body {  
23   font-family: "Segoe UI", Arial, "Helvetica Neue", Helvetica, sans-serif;  
24   background-color: $bg;  
25 }  
26  
27 h1, h2 {  
28   color: white;  
29 }  
30  
31 blockquote {  
32   font-style: italic;  
33   @include paraTelasPequenas(420) {  
34     margin-left: 10px;  
35   }  
36 }  
37  
38 // Header  
39 .masthead {  
40   display: flex;  
41   justify-content: space-between;  
42   max-width: 980px;  
43   margin: auto;  
44   padding: 10px;  
45   background: $b;  
46   @include paraTelasPequenas(700) {  
47     display: block;  
48     text-align: center;  
49   }  
50 }  
51  
52 .logo {  
53   padding: 10px 0 10px 10px;  
54   color: white;
```





```
59     font-size: 1.5em;
60 }
61 @include paraTelasPequenas(420) {
-- }
64 }
65
66 //Placeholder
67 %highlight-section {
68 /*border: white 1px solid;
69 border-radius: 3px;
70 background: rgba(white, .1);*/
71 background: rgba(black,.2);
72 border-radius: 3px;
73 }
74
75
76 // Campo de busca
77 .search {
78     @extend %highlight-section;
79     display: flex;
80     align-items: center;
81     justify-content: center;
82     text-align: center;
83     color: white;
84     font-size: .9em;
85     padding: 10px;
86     .field {
87         width: 50%;
88         margin: 0 5px;
89         padding: 7px;
90         @include paraTelasPequenas(420) {
91             width: 70%;
92         }
93     }
94     button {
95         @include paraTelasPequenas(420) {
96             width: 90%;
97             margin-top: 10px;
98         }
99     }
100 }
```





```
105    width: 100%;  
106    margin-bottom: 10px;  
107  
108    }  
109  
110    }  
111  
112  
113 // Elementos de formulário  
114 input, textarea {  
115     font-family: "Segoe UI", Arial, "Helvetica Neue", Helvetica, sans-serif;  
116     padding: 10px;  
117     border: none;  
118     background: rgba(white,.8);  
119     border-radius: 2px;  
120     box-shadow: inset 0 1px 1px rgba(black,.5);  
121 }  
122  
123 button {  
124     border: none;  
125     padding: 7px 10px;  
126     background: #333;  
127     border-radius: 3px;  
128     color: white;  
129     font-family: "Segoe UI", Arial, "Helvetica Neue", Helvetica, sans-serif;  
130     cursor: pointer;  
131 }
```



BOX 2. Mixins

Um mixin é uma diretiva que permite a definição de múltiplas regras que serão traduzidas para o CSS dependendo dos argumentos enviados dinamicamente por parâmetro. Pense num mixin como uma função que permite reusar estilo ao longo da folha de estilos sem precisar recorrer a recursos estáticos do CSS em si.



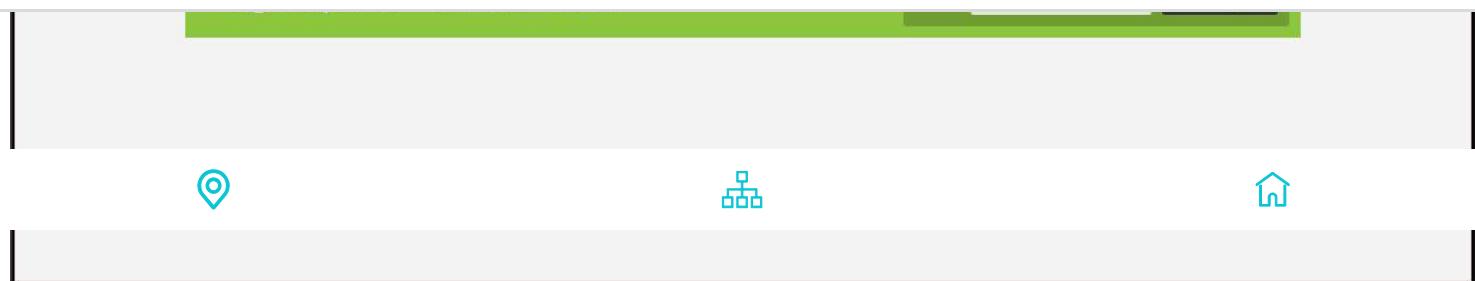


Figura 2. Tela com cabeçalho responsivo.



Figura 3. Tela com cabeçalho responsivo em dispositivo pequeno.

Vamos incluir agora o restante do conteúdo HTML da página, que se divide entre barra de navegação (`<nav>`), corpo da página (`<main>`) e rodapé (`<footer>`). Logo, inclua o conteúdo da **Listagem 7** após a declaração do cabeçalho recém-criado. Nela, podemos perceber a inclusão das seguintes tags principais:



“navigation” que debatemos antes.

- **main:** guarda todo o conteúdo principal da página (com a `role` de “main”), que inclui



tag `<article>` com o texto do post (no qual também inserimos um `<aside>` com `role` “complementary” para complementar o conteúdo do post).

- **form:** está inserido dentro do `main`, mas traz configurações específicas do formulário de contato (`role` “form”), com uma segunda tag `<header>` (dessa vez interna a uma seção), bem como os campos de `input`, `textareta` e botão de `submit`.
- **footer:** através da `role` “contentinfo”, o rodapé traz apenas um texto de *copyright* e mais uma lista de links de menu tal como tivemos no `header`.

Listagem 7. Restante do código HTML de navegação, corpo e rodapé.

```
1 <nav class="main-nav" role="navigation">
2   <ul class="nav-container">
3     <li><a href="#">Link 1</a></li>
4     <li><a href="#">Link 2</a></li>
5     <li><a href="#">Link 3</a></li>
6     <li><a href="#">Link 4</a></li>
7   </ul>
8 </nav>
9 <main class="main-container" role="main">
10   <h1>Criando conteúdo em HTML5</h1>
11   <p>HTML5 é uma linguagem de marcação de hipertexto, muito utilizada na...
12   <article class="article-container flex-container">
13     <section class="main-content">
14       <header>
15         <h1>O elemento <code><main></code></h1>
16       </header>
17       <p>Por definição:</p>
18       <blockquote>
19         <p>O Elemento (<code><main></code>) representa...
20       </p>
21     </blockquote>
22   </section>
```





```
28 |     <div class="contact-form" role="form">
29 |         <header>
30 |             <h2>Mande-nos sua sugestão/dúvida/questão</h2>
-->
33 |             <div class="flex-container">
34 |                 <label class="label-col">Nome:</label>
35 |                 <input type="text" class="field name" id="name">
36 |             </label>
37 |             <label class="label-col">Email:</label>
38 |             <input type="email" class="field email" id="email">
39 |         </label>
40 |     </div>
41 |     <label for="comentario">Comentário:</label>
42 |
43 |     <textarea class="comments" id="comentario" cols="50" required>
44 |         <button>Enviar</button>
45 |     </form>
46 | </div>
47 | <footer class="main-footer" role="contentinfo">
48 |     <p>Copyright © 2015 | Todos os direitos reservados. </p>
49 |     <ul class="nav-container" role="navigation">
50 |         <li><a href="#">Rodapé Link 1</a></li>
51 |         <li><a href="#">Rodapé Link 2</a></li>
52 |         <li><a href="#">Rodapé Link 3</a></li>
53 |         <li><a href="#">Rodapé Link 4</a></li>
54 |         <li><a href="#">Rodapé Link 5</a></li>
55 |     </ul>
56 | </footer>
57 | </main>
```

Agora precisamos criar as regras Sass para lidar com o estilo dos novos componentes. Como se trata de muito conteúdo, vamos dividir em duas partes: **Listagens 8 e 9**. Na primeira temos as definições divididas da seguinte forma:

- **Navegação:** mapeia o CSS para a tag `<nav>` definindo o tamanho máximo de largura, cor de plano de fundo (veja que nesta usamos o valor hexadecimal da cor `inline`, isto





reflete na função `rgba()` do Sass definida nas propriedades `box-shadow` e `border-bottom` do link (`<a>`). Essa função recebe quatro argumentos: as cores `red`, `green` e



campo (valor deve estar entre 0 e 1).

- **Container principal:** define propriedades gerais para o corpo da página. Veja que aqui fazemos uso da variável global `&` criada no início do arquivo para configurar a cor da `border-bottom` do mesmo. Destaque para o elemento `<h1>` que recebe suas propriedades de estilo, aplicando-as a todos os `h1's` da página. Isso se dá através do código `&>h1` (o caractere `&` quer dizer “todo o documento” no Sass).
- **Article:** configura as propriedades do artigo que incluem margem, cor de fundo, padding e bordas.

Listagem 8. Código Sass para navegação, corpo e artigo.

```
1 // Navegação
2 .main-nav {
3     max-width: 980px;
4     margin: auto;
5     padding: 10px 5px;
6     border-bottom: rgba(black,.2) 1px solid;
7     background: #49c5bf;
8     @include paraTelasPequenas(420) {
9         padding: 5px 0;
10    }
11 }
12
13 // Todas as navegações
14 .nav-container {
15     display: flex;
16     justify-content: center;
17     list-style-type: none;
18     margin: 0;
19     padding: 0;
20     @include paraTelasPequenas(420) {
21         flex-wrap: wrap;
```





```
26     margin: 0 5px;
27     text-align: center;
28     @include paraTelasPequenas(420) {
29         flex-grow: 1;
30         flex-basis: 45%;
31         margin: 5px;
32     }
33 }
34 }
35 a {
36     @extend %highlight-section;
37     display: flex;
38     justify-content: center;
39     align-items: center;
40     width: 100%;
41     padding: 10px;
42     color: white;
43     text-decoration: none;
44     font-size: 1.3em;
45     box-shadow: inset 0 1px 2px rgba(black,.2);
46     border-bottom: rgba(white,.6) 1px solid;
47 }
48 }
49
50 // Container Principal
51 .main-container {
52     max-width: 980px;
53     margin: auto;
54     padding: 40px 40px 20px;
55     background: #2a2a2a;
56     color: rgba(white,.8);
57     border-bottom: $y 40px solid;
58     @include paraTelasPequenas(420) {
59         padding: 20px;
60         line-height: 1.5;
61     }
62 //Main Headings
63 &>h1 {
64     margin-top: 0;
65     padding-bottom: 10px;
66     border-bottom: $y 3px solid;
67     @include paraTelasPequenas(420) {
```





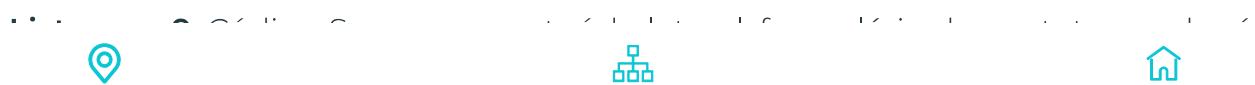
```
72
73 // Article
74 .article-container {
75   position: relative;
76   width: 100px;
77   background: rgba(white,.05);
78   border-radius: 2px;
79   border: rgba(black,.2) 10px solid;
80   box-shadow: 0 5px 15px rgba(black,.3);
81 }
82
83 // Conteúdo principal da página
84 .main-content {
85   width: 75%;
86   margin-right: 10px;
87   padding: 10px;
88 @include paraTelasPequenas(600) {
89   width: 100%;
90 }
91 h1 {
92   margin: 0;
93   padding-bottom: 10px;
94   border-bottom: $g 3px solid;
95 }
96 }
```



Para a segunda listagem certifique-se de incluir seu conteúdo no final do arquivo `estilos.scss`. Nela podemos ver recursos gerais para os demais componentes da página, a saber:

- **Conteúdo lateral:** configura o estilo dos títulos de cabeçalho, links, listas ordenadas e parágrafo que possam existir no corpo do post.
- **Formulário de Contato:** cria uma espécie de delimitação ao redor da div de formulário, configurando propriedades de estilo específicas para os campos de botão, labels, input, textarea desse formulário. Estas sobrescreverão as demais criadas antes para campos de formulário.





```
1 // Conteúdo lateral
2 .side-content {
3     width: 25%;
4     padding: 10px;
5     font-size: .8em;
6     border: $b 2px dotted;
7     border-left: none;
8     border-right: none;
9     @include paraTelasPequenas(600) {
10         width: 100%;
11         margin-top: 12px;
12     }
13     h2 {
14         margin: 0;
15     }
16     ol {
17         padding-left: 20px;
18     }
19     a {
20         color: #eee;
21     }
22     p {
23         @include paraTelasPequenas(420) {
24             font-size: 1.2em;
25         }
26     }
27 }
28
29 // Formulário de Contato
30 .contact-form {
31     width: 540px;
32     margin: 40px auto;
33     padding: 20px;
34     border: rgba(black,.2) 10px solid;
35     box-shadow: 0 5px 15px rgba(black,.3);
36     @include paraTelasPequenas(600) {
37         width: 100%;
```





```
42     border-bottom: $r 3px solid;
43 }
44 label, button {
45   border-radius: 10px;
46   padding: 10px;
47   input, textarea {
48     margin-top: 5px;
49   }
50   .comments {
51     height: 100px;
52   }
53   button {
54     font-size: 1.3em;
55     background: $b;
56     @include paraTelasPequenas(420) {
57       width: 100%;
58     }
59   }
60   .flex-container {
61     justify-content: space-between;
62     @include paraTelasPequenas(600) {
63       display: flex;
64     }
65     @include paraTelasPequenas(400) {
66       display: block;
67     }
68   }
69   .label-col {
70     width: 48%;
71     @include paraTelasPequenas(400) {
72       width: 100%;
73     }
74   }
75 }

76

77

78 // Rodapé
79 .main-footer {
80   color: white;
81   padding: 10px;
82   border-top: $y 2px dotted;
83   font-size: 0.7em;
```





```
88 }  
89  
90 // Classes utilitárias  
--  
93 @include paraTelasPequenas(600) {  
94     display: block;  
95 }  
96 }
```



Após isso, o leitor já pode salvar todos os documentos e verificar o resultado no browser.

Na **Figura 4** temos o resultado do site num navegador em modo desktop, e na **Figura 5** a mesma página visualizada a partir de um iPhone 6 Plus. Veja como os elementos se rearranjam em detrimento da diminuição de espaço disponível.





Link 1

Link 2

Link 3

Link 4



HTML5 é uma linguagem de marcação de hipertexto, muito utilizada na web...

O elemento <main>

Por definição:

O Elemento (<main>) representa...

Quer se tornar um programador?

Provavelmente você terá um longo caminho, mas pode fazer uso do nosso conteúdo para te ajudar.

Mande-nos sua sugestão/dúvida/questão

Nome:

Email:

Comentário:

Enviar

Copyright © 2015 | Todos os direitos reservados.

Rodapé Link 1

Rodapé Link 2

Rodapé Link 3

Rodapé Link 4

Rodapé Link 5

Figura 4. Tela final visualizada de um browser desktop.





Figura 5. Tela final visualizada de um iPhone 6 Plus.

Grids CSS

Uma grid é um conjunto de guias visuais (verticais, horizontais ou ambas) que ajudam a definir onde os elementos podem ser alocados. Uma vez que eles tenham sido alocados, alcançamos o conceito de **layout**. Uma das maiores vantagens de usar grids é que seus elementos internos terão um fluxo harmonioso ao longo das páginas, incrementando a





Muitos frameworks de componentes já usam esse conceito, como o Bootstrap por exemplo, que mantém o container com uma largura de no máximo 980px e divide seu espaço



O conceito se baseia em definir para cada grid as “linhas” (rows) que, por sua vez, serão constituídas de colunas (columns) em quantidade máxima de 12. Vejamos o exemplo exposto pela **Listagem 10**. No início dela configuramos as propriedades de box-sizing para definir uma borda padrão ao documento como um todo.

A classe “container-12” deve ser a primeira e mais ao topo da `div` que vai conter todo o conteúdo principal. Veja que definimos sua largura fixa em 980px (se o leitor não quiser fazer uso do `@media` para quando esta classe for executada em dispositivos menores pode definir essa largura como máxima permitida - “max-width” – e a largura em si (“width”) com o valor de 100%).

Em seguida, fazemos uso do Sass para imprimir o `float` à esquerda e uma margem para todas as classes de grid, da 1 até a 12. Note que cada grid tem seu valor prefixado de acordo com as dimensões de tamanho máximo da página (de 60px – valor mínimo – até 940px – valor máximo).

Listagem 10. Código de exemplo para sistema de grids.

```
1  *,
2  *:before,
3  *:after {
4      box-sizing: border-box;
5  }
6 // Container
7 .container-12 {
8     width: 980px;
9     padding: 0 10px;
10    margin: auto;
```





```
16     margin: 0 10px;  
17 }  
18 }
```



```
--  
21 .grid-1 {  
22     width: 60px;  
23 }  
24 .grid-2 {  
25     width: 140px;  
26 }  
27 .grid-3 {  
28     width: 220px;  
29 }  
30 .grid-4 {  
31     width: 300px;  
32 }  
33 .grid-5 {  
34     width: 380px;  
35 }  
36 .grid-6 {  
37     width: 460px;  
38 }  
39 .grid-7 {  
40     width: 540px;  
41 }  
42 .grid-8 {  
43     width: 620px;  
44 }  
45 .grid-9 {  
46     width: 700px;  
47 }  
48 .grid-10 {  
49     width: 780px;  
50 }  
51 .grid-11 {  
52     width: 860px;  
53 }  
54 .grid-12 {  
55     width: 940px;  
56 }
```

7





```
62     content: '';
63     display: table;
64   }
-->
67   }
68 }
69 // Usa linhas para aninhar os containers
70 .row {
71   margin-bottom: 10px;
72   &:last-of-type {
73     margin-bottom: 0;
74   }
75 }
76 // Legado IE
77 .clear {
78   zoom: 1;
79 }
```



Além dessa configuração feita fixamente em pixels, podemos implementar o modelo todo baseado em percentagem. Para isso, precisamos nos assegurar que a largura máxima do container também esteja em percentagem, bem como as linhas e as demais divisórias do documento. Veja na **Listagem 11** como ficaria o mesmo código com as alterações percentuais.

Para calcular esses valores, o leitor pode fazer uso de uma fórmula bem simples: $(alvo / contexto) \times 100 = resultado\%$. No nosso exemplo, o alvo seria o valor de cada propriedade em pixels e o contexto seria o valor máximo de largura, ou seja, 980px.

Listagem 11. Código de exemplo para sistema de grids com percentual.

```
1 // Container
2 .container-12 {
3   width: 100%;
```





```
9  .grid {  
10     &-1, &-2, &-3, &-4, &-5, &-6, &-7, &-8, &-9, &-10, &-11, &-12 {  
11         float: left;  
--  
14     }  
15 // Grid >> 12 colunas  
16 .container-12 {  
17     .grid-1 {  
18         width: 6.12%;  
19     }  
20     .grid-2 {  
21         width: 14.29%;  
22     }  
23     .grid-3 {  
24         width: 22.45%;  
25     }  
26     .grid-4 {  
27         width: 30.61%;  
28     }  
29     .grid-5 {  
30         width: 38.78%;  
31     }  
32     .grid-6 {  
33         width: 46.94%;  
34     }  
35     .grid-7 {  
36         width: 55.10%;  
37     }  
38     .grid-8 {  
39         width: 63.27%;  
40     }  
41     .grid-9 {  
42         width: 71.43%;  
43     }  
44     .grid-10 {  
45         width: 79.59%;  
46     }  
47     .grid-11 {  
48         width: 87.76%;  
49     }  
50     .grid-12 {
```





```
55 .clear,  
56 .row {  
57   &:before, &:after {  
58     content: " ";  
59     display: block;  
60     width: 0;  
61     height: 0;  
62     &:after {  
63       clear: both;  
64     }  
65   }  
66   // Usa linhas para aninhar os containers  
67   .row {  
68     margin-bottom: 10px;  
69     &:last-of-type {  
70       margin-bottom: 0;  
71     }  
72   }  
73   // Legado IE  
74   .clear {  
75     zoom: 1;
```



Tipografia

A tipografia (seleção de fontes) de um site é uma das partes mais importantes da definição responsiva de um design, já que precisamos também considerar como nossos textos, títulos, subtítulos, etc. irão se comportar em diferentes cenários. Dentro dessa questão, muitas perguntas surgem como “deve-se usar pixels, ems, ou rems?”, “que scale devo aplicar?” ou “qual a melhor fonte para cada seção?”.

Dentre outras decisões importantes, a definição do tamanho base da fonte, bem como seu ratio precisam ser tomadas no início da criação da página, para se certificar de que as demais sigam o mesmo ritmo.

Na seção **Links** você encontra a URL para o site *Modular Scale*, uma ferramenta poderosa



padrão para a base, em pixels, e clicamos no botão *Table* que mostra uma relação dos valores de fonte em pixels (primeira coluna), ems (segunda coluna) e o possível valor se o



O leitor pode ir aumentando ou diminuindo tais valores e ver como o texto se comporta até que tenha uma definição final de qual usará. O mesmo vale para o `ratio`.

	Table	Text	px	ems
ms(6)	8.472px	4.236em	0.53em @ 16	
ms(5)	6.213px	3.107em	0.388em @ 16	
ms(4)	5.236px	2.618em	0.327em @ 16	
ms(3)	3.84px	1.92em	0.24em @ 16	
ms(2)	3.236px	1.618em	0.202em @ 16	
ms(1)	2.373px	1.187em	0.148em @ 16	
ms(0)	2px	1em	0.125em @ 16	
ms(-1)	1.236px	0.618em	0.077em @ 16	
ms(-2)	0.764px	0.382em	0.048em @ 16	
ms(-3)	0.472px	0.236em	0.03em @ 16	
ms(-4)	0.292px	0.146em	0.018em @ 16	
ms(-5)	0.18px	0.09em	0.011em @ 16	
ms(-6)	0.111px	0.056em	0.007em @ 16	

\$ms-base: 2px, 2000px;
\$ms-ratio: 1.618;

Figura 6. Definindo scales para fontes na Modular Scale.

Na **Listagem 12** temos um exemplo de página HTML gerada com alguns cabeçalhos de título (`<h1>` `<h6>`) para que possamos entender a aplicação dos valores `base/ratio`.

Veja que criamos um segundo arquivo de estilo SCSS para garantir que as regras não se misturem com as definidas no primeiro exemplo.

Na **Listagem 13** você encontra o código Sass para aplicar o estilo desejado aos referidos componentes. Note que implementamos o mesmo conceito de função mixin para receber o valor e converter as fontes para telas maiores.

No estilo do `h1` definimos a base da fonte como `16px` e o `ratio` como `1.4` e nos



**Listagem 12.** HTML para aplicação de base/ratio na fonte.

```
2 <html>
3
4 <head>
5     <meta charset="utf-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width, initial-scale=1">
8     <title>Design Responsivo com HTML5 & CSS3</title>
9     <link rel="stylesheet" href="style/css/estilos2.css">
10 </head>
11
12 <body>
13     <h1>Exemplo de tipografia para Design Responsivo</h1>
14     <blockquote>
15         <p>"Você não pode juntar os pontos olhando para frente; você pode coi
16             <p>— Steve Jobs</p>
17         </blockquote>
18         <h2>Typografia - Fonte H2</h2>
19         <p>Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, conse
20             <h3>Exemplo H3</h3>
21             <p>Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, conse
22     </body>
23
24 </html>
```

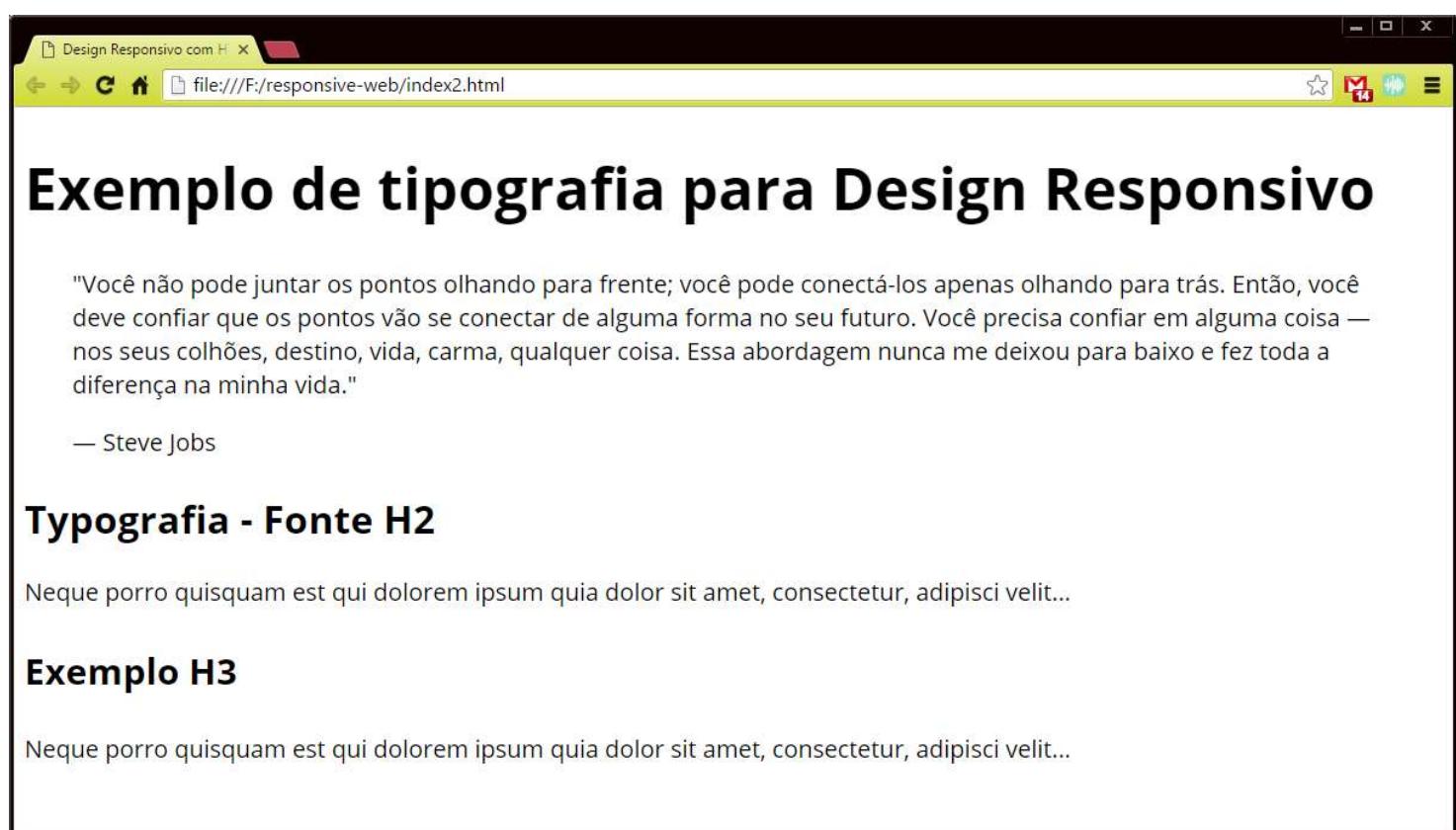
Listagem 13. Código Sass para aplicar base/ratio nos títulos.

```
1 //Mobile-first Media Query Mixin
2 @Mixin paraTelasGrandes($media) {
3     @media (min-width: $media/16+em) {
4         @content;
5     }
6 }
7 body {
8     font: 16px/1.4 Open Sans Arial "Helvetica Neue" Helvetica sans-serif
9 }
```





```
13 | h1 {  
14 |     font-size: 2.454em;  
15 | }  
--  
18 | }  
19 | h3 {  
20 |     font-size: 1.517em;  
21 | }
```



Exemplo de tipografia para Design Responsivo

"Você não pode juntar os pontos olhando para frente; você pode conectá-los apenas olhando para trás. Então, você deve confiar que os pontos vão se conectar de alguma forma no seu futuro. Você precisa confiar em alguma coisa — nos seus colhões, destino, vida, carma, qualquer coisa. Essa abordagem nunca me deixou para baixo e fez toda a diferença na minha vida."

— Steve Jobs

Typografia - Fonte H2

Nequer porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit...

Exemplo H3

Nequer porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit...

Figura 7. Tela com tipografia aplicada via Modular Scale.

Definir um design responsivo é tarefa custosa e muitas vezes envolve vários riscos como a falta de maturidade dos profissionais envolvidos em sua construção, falta de tempo e organização para definir os escopos, falta de conhecimento sobre que tipos de websites receberão aquele estilo, bem como o não entendimento da quantidade de mudanças que esse mesmo site irá sofrer com o decorrer do tempo.





futuros sites/sistemas. Outro enfoque importante nessa abordagem é o SEO.

Têm bom design que se sempre implica em melhorias no SEO de uma página. Todavia, se o



CSS final poderá ficar gigante e não performático.

Em relação às bibliotecas de terceiros (*third parties*) é preciso ter muito cuidado com seu uso. Por um lado, elas podem acrescentar em muito ao design e facilitar certas tarefas que demorariam muito se feitas manualmente, entretanto, por outro, podem importar uma carga muito alta de código que, em sua grande maioria, não será usada, logo, perdemos em performance mais uma vez.

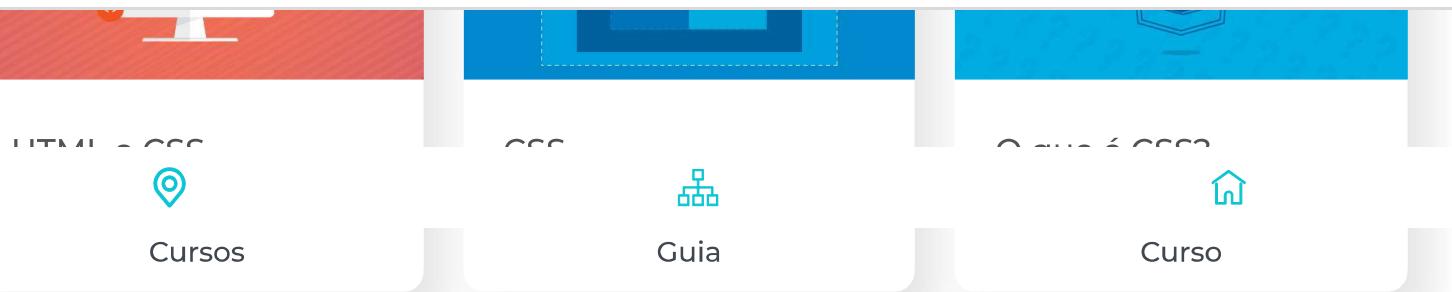
Tudo é uma questão de conhecimento da biblioteca e da existência de outras substitutas que possam fornecer o mesmo resultado, com melhor produtividade e usando mais recursos da mesma. O único jeito de alcançar isso é experienciando.

Links

- [Ruby](#)
- [LibSass](#)
- [RubyInstaller](#)
- [Documentation](#)
- [Modular Scale](#)
- [HTML](#)
- [Criando um aplicativo Desktop](#)

Confira também





HTML e CSS CSS Guia Curso

Cursos Guia Curso

Tecnologias:[CSS](#) [HTML](#) [Sass](#)[Marcar como concluído](#)  [Anotar](#) 

Por **Fabricio**
Em 2016

Suporte ao aluno - Deixe a sua dúvida.



23

[Cursos](#)[Artigos](#)[Fale conosco](#)[Trabalhe conosco](#)[Assinatura para empresas](#)

Hospedagem web por Porta 80 Web Hosting

