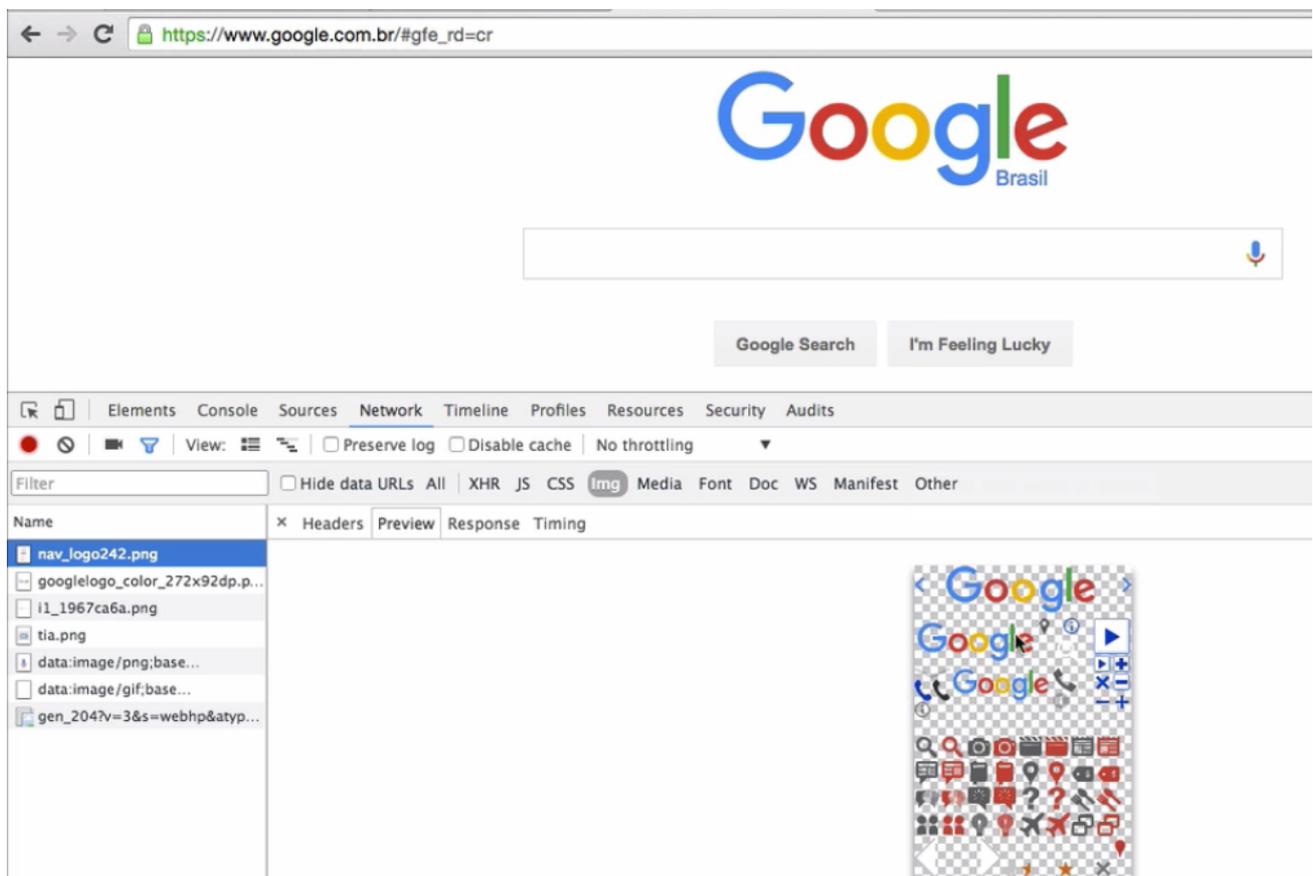


Transcrição das aulas

Já realizamos diversas modificações, mas ainda temos diversos aspectos com os quais nos preocupamos. Temos diversos ícones e imagens em nosso site e todas elas estão em arquivos separados. Vamos observar isso a partir da pasta "img" que está em "Site > assets > img". Nesta pasta temos imagens de diversos tipos, nela temos 53 itens, ou seja, 53 *requests* que são realizados na *home*.

Para evitar o encavalamento, tópico que já analisamos e discutimos anteriormente, uma opção é concatenar as imagens da mesma maneira que fizemos com o *css*. Entretanto, concatenar imagens não é tão simples, mas é possível. A concatenação de imagens também é chamada de *sprite*.

Um ótimo exemplo sobre isso é o site do *Google*. Vamos dar um "F12" e observar a aba *network*, vamos selecionar o *img*. Se clicarmos no primeiro arquivo conseguimos observar o que é o *sprite*:



É uma imagem que contém diversos ícones. Isso é o que chamamos de *sprite*.

Ao em vez de fazer um *requests* para cada uma dessas imagens que estão contidas nessa única imagem, tudo é juntado em um mesmo *png* e que baixa todos os ícones uma única vez. É interessante perceber que se tivermos vários ícones parecidos em um arquivo só o seu tamanho é menor do que se esses ícones estivessem separados.

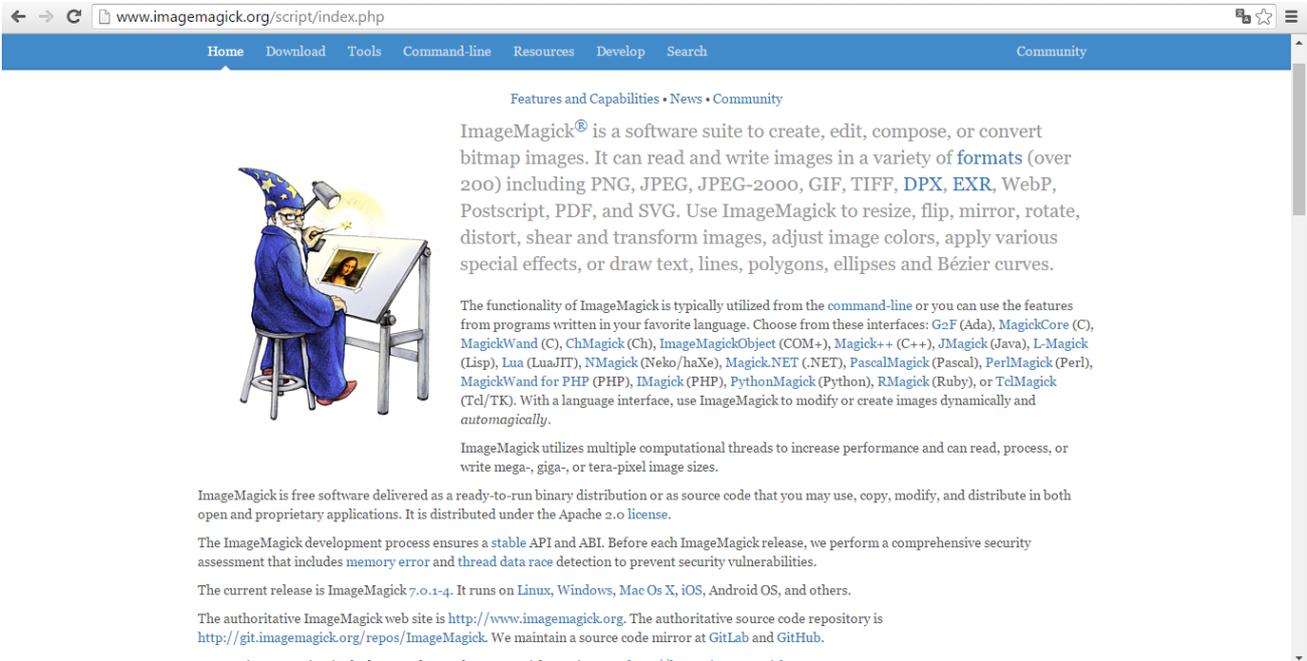
E como começamos a fazer isso?

Na realidade, quando observamos um site ele possui sua distribuição de ícones organizada, eles estão em diferentes pontos da página, embora, como no caso do *google*, estejam todos em um arquivo só. Organizamos isso utilizando técnicas de *sprite* com *css*. O que fazemos é redimensionar o *div* e reposicionamos o *background* para exibir apenas um pedaço da imagem. Isto é, temos uma imagem grande de *background*, mas o *div*, como é pequeno, posicionamos ele de

maneira a mostrar apenas o pedacinho do que queremos, por exemplo, uma "setinha", dando a impressão de que temos apenas um ícone.

Vamos fazer isso na prática!

A primeira coisa que precisamos fazer é criar essa imagem e desenhar ela. Você pode fazer isso utilizando o *photoshop* ou, o ideal é automatizar essa tarefa. Uma ferramenta que aconselhamos para automatizar imagens é o *Image Magick*, que está no site www.imagemagick.org/.



ImageMagick® is a software suite to create, edit, compose, or convert bitmap images. It can read and write images in a variety of **formats** (over 200) including PNG, JPEG, JPEG-2000, GIF, TIFF, DPX, EXR, WebP, Postscript, PDF, and SVG. Use ImageMagick to resize, flip, mirror, rotate, distort, shear and transform images, adjust image colors, apply various special effects, or draw text, lines, polygons, ellipses and Bézier curves.

The functionality of ImageMagick is typically utilized from the **command-line** or you can use the features from programs written in your favorite language. Choose from these interfaces: G2F (Ada), MagickCore (C), MagickWand (C), ChMagick (Ch), ImageMagickObject (COM+), Magick++ (C++), JMagick (Java), L-Magick (Lisp), Lua (LuaJIT), NMagick (Neko/haXe), Magick.NET (.NET), PascalMagick (Pascal), PerlMagick (Perl), MagickWand for PHP (PHP), IMagick (PHP), PythonMagick (Python), RMagick (Ruby), or TdMagick (Tcl/TK). With a language interface, use ImageMagick to modify or create images dynamically and *automagically*.

ImageMagick utilizes multiple computational threads to increase performance and can read, process, or write mega-, giga-, or tera-pixel image sizes.

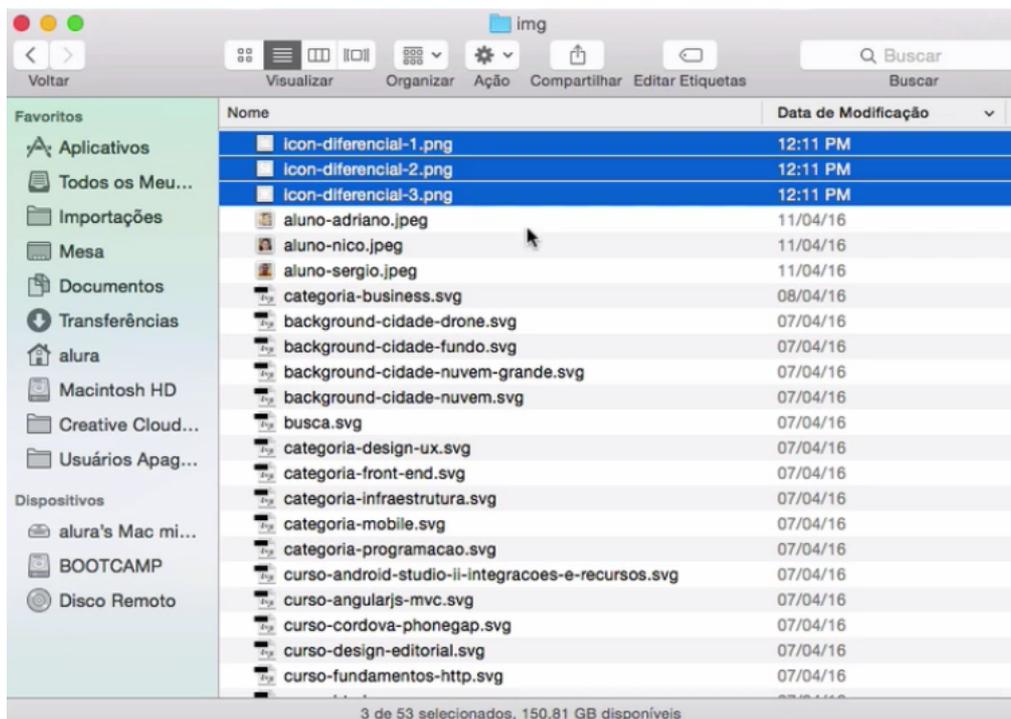
ImageMagick is free software delivered as a ready-to-run binary distribution or as source code that you may use, copy, modify, and distribute in both open and proprietary applications. It is distributed under the Apache 2.0 license.

The ImageMagick development process ensures a **stable** API and ABI. Before each ImageMagick release, we perform a comprehensive security assessment that includes **memory error** and **thread data race** detection to prevent security vulnerabilities.

The current release is ImageMagick 7.0.1-4. It runs on Linux, Windows, Mac OS X, iOS, Android OS, and others.

The authoritative ImageMagick web site is <http://www.imagemagick.org>. The authoritative source code repository is <http://git.imagemagick.org/repos/ImageMagick>. We maintain a source code mirror at [GitHub](https://github.com/ImageMagick/ImageMagick) and [GitLab](https://gitlab.com/ImageMagick/ImageMagick).

A ideia é que podemos passar um comando, usaremos o comando *convert* que é cheio de opções. Ele faz vários tipos de edição de imagem só que na linha de comando. Portanto, ele é muito útil quando se trata de automatização. Vamos instalar o *Image Magick* e como temos vários ícones distintos, vamos tentar criar uma *sprite* para diminuir os *requests*. Vamos pegar três ícones que diferenciam as funcionalidades do Alura:



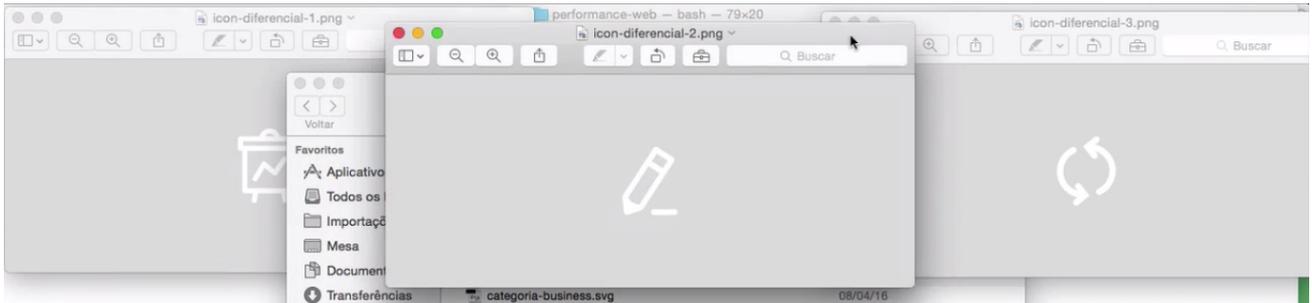
Esses três ícones são *png* e possuem os respectivos nomes: *icon-diferencial-1*, *icon-diferencial-2* e *icon-diferencial-3*.

A ideia é que juntemos os três arquivos em um único arquivo. Vamos para a linha de comando, após instalarmos a ferramenta, e usaremos o comando `convert` e passaremos as imagens que queremos transformar, no caso, todos os `png` da seguinte pasta: `site/assets/img/*.png` e como queremos transformar isso em um único arquivo. Para fazer isso, utilizamos o comando `append`, que pega as imagens e coloca umas em baixo das outras. Por fim, jogamos a imagem final na pasta `site/assets/img/diferenciais.png`. Teremos na linha de comando o seguinte:

```
convert site/assets/img/*.png -append site/assets/img/diferenciais.png
```

Damos um "Enter" nisso e veremos se funcionou.

Bom, os ícones que queremos juntar são os seguintes:



Geramos um diferencial `png` que possui os três ícones em uma mesma imagem e um em baixo do outro, essa distribuição horizontal é o padrão, mas a orientação pode ser modificada. Repare que cada um dos `png` separados possuíam 3 KB cada enquanto nossa imagem com os três possui 4 KB. Ainda, otimizaremos essa imagem tornando ela ainda menor.

E agora, como usamos esse arquivo com as três imagens no nosso site?

Vamos observar o código fonte para perceber como ele é exibido. Vamos olhar o `html` e na parte dos diferenciais temos basicamente que cada um dos ícones é um `span` e cada um deles está nomeado de uma maneira distinta:

```

179
180 <div class="home-diferenciais-item">
181   <span class="home-diferenciais-icone icone-carreira"></span>
182   <div class="home-diferenciais-item-texto">
183     <h3 class="home-diferenciais-item-nome">Sua carreira em alta</
184     <p class="home-diferenciais-item-descricao">Tudo que você prec
185   </div>
186 </div>
187
188 <div class="home-diferenciais-item">
189   <span class="home-diferenciais-icone icone-cresca"></span>
190   <div class="home-diferenciais-item-texto">
191     <h3 class="home-diferenciais-item-nome">Você aprende na prátic
192     <p class="home-diferenciais-item-descricao">No Alura você faz
193   </div>
194 </div>

```

Se é um `span` isso significa que não tem nada dentro dele e estamos usando isso dentro do `css`. Abrimos o arquivo `"home-diferenciais.css"` e localizamos a parte que fala dos ícones:

```
home-diferenciais.css x
60  .icone-cresca:before {
61      background-image: url("../img/icon-diferencial-1.png");
62  }
63  .icone-carreira:before {
64      background-image: url("../img/icon-diferencial-2.png");
65  }
66  .icone-impacto:before {
67      background-image: url("../img/icon-diferencial-3.png");
68  }
69
70  /* fim dos icones */
71
72
73
```

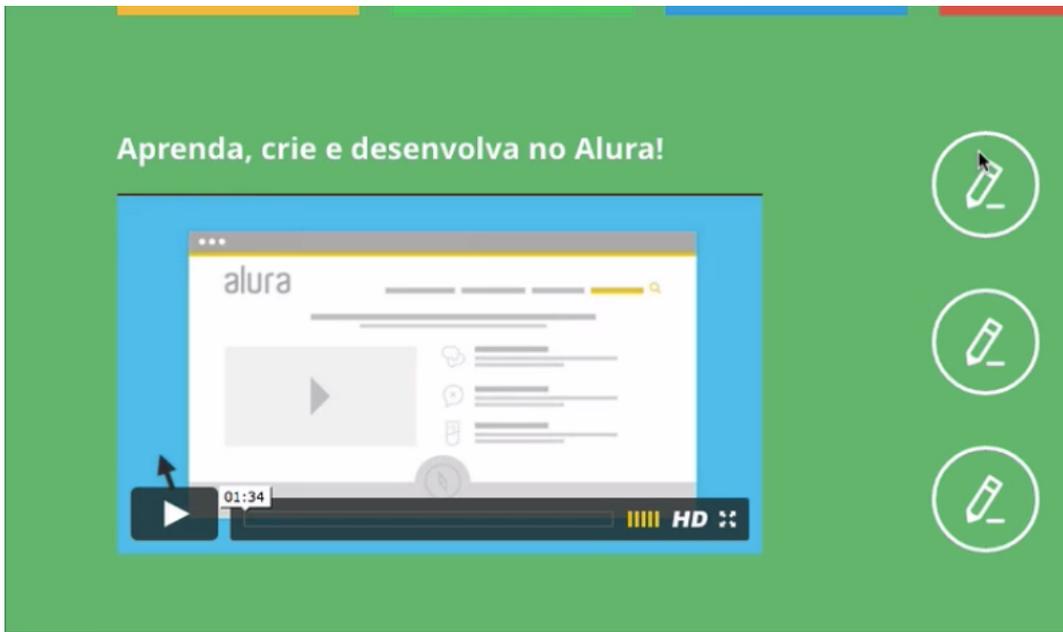
Nessa parte estão descritas as regras das imagens, as bordas arredondadas e outros detalhes... O que queremos mostrar é que cada um daqueles ícones estão sendo diferenciados com um *background image* diferente.

O que vamos fazer?

Vamos parar de usar os arquivos diferentes e vamos utilizar o arquivo novo e único, vamos alterar, então, o nome do arquivo para `../img/diferenciais.png` e colocar isso para cima e apagar os nomes que diferenciavam as imagens.

```
61
62  .icone-cresca:before {
63  }
64  .icone-carreira:before {
65  }
66  .icone-impacto:before {
67  }
68  }
69
70  /* fim dos icones */
71
72
```

A ideia é que vamos usar o mesmo *background*, um único arquivo, nos três ícones. Vamos dar um *refresh* para ver o que acontece:



Vemos que temos o mesmo ítem nos três espaços que eram reservados para ícones diferentes. Vamos dar um "F12" e ficar na aba *Elements*. O que acontece é que o arquivo "diferenciais.png" é muito longo e como ele tem três ícones o que ocorre é que vemos um ícone apenas de cada vez. O que queremos, na verdade, é que em cada uma das opções seja mostrado um ícone diferente. Fazemos isso deslocando a imagem do *background*, da *sprite*, isto é, deslocamos alguns pixels. Vamos observar o terminal, por exemplo, podemos pegar `home-diferenciais.css` e colocar um `background-position` e através de um `-y` podemos controlar quantos pixels queremos mostrar. Perceba que conforme vamos alterando os `pixels`, estamos alterando qual o ítem que ocupará esse espaço:

Tudo que você precisa para ser um melhor profissional em tecnologia e sua carreira decolar.

Você aprende na prática

No Alura você faz diversos exercícios que estimulam a aprender de verdade com projetos reais.

O conteúdo que você precisa

Já são 208 cursos e com novos lançamentos todos os meses, além de atualizações e

```

<html lang="pt-BR" class="home-fundo-animado">
<body>
  <div class="home-fundo">
  <div class="home-diferenciais">
    <div class="container">
      <div class="home-diferenciais-video">
      <div class="home-diferenciais-list">
        <div class="home-diferenciais-item">
          <span class="home-diferenciais-icone icone-carreira">
            ::before
          </span>
          <div class="home-diferenciais-item-texto">
          </div>
        <div class="home-diferenciais-item">
          <span class="home-diferenciais-icone icone-cresca">
            ::before
          </span>
          <div class="home-diferenciais-item-texto">
          </div>
        <div class="home-diferenciais-item">
          <span class="home-diferenciais-icone icone-estudo">
            ::before
          </span>
          <div class="home-diferenciais-item-texto">
          </div>
        </div>
      </div>
    </div>
  </div>
</body>
  
```

```

.icone-cresca:before {
  background-position-y: -34px;
}
  
```

Se soubermos o tamanho do ícone basta irmos mexendo que conseguimos exibir um ícone ou outro. Se queremos exibir o ícone que está mais na ponta temos que acrescentar `-80 px` porque é menos os outros dois - cada um tem "40 pixels" - e assim por diante. Podemos utilizar, ainda, alguns atalhos: `top` para fazer referência ao primeiro ícone, `bottom` para mostrar o último e `center` para mostrar o do meio. Ficaremos, por exemplo, com `background-positions-y: top`. Se tivermos muitos ícones, entretanto, esse atalho não funciona, teremos que usar as coordenadas.

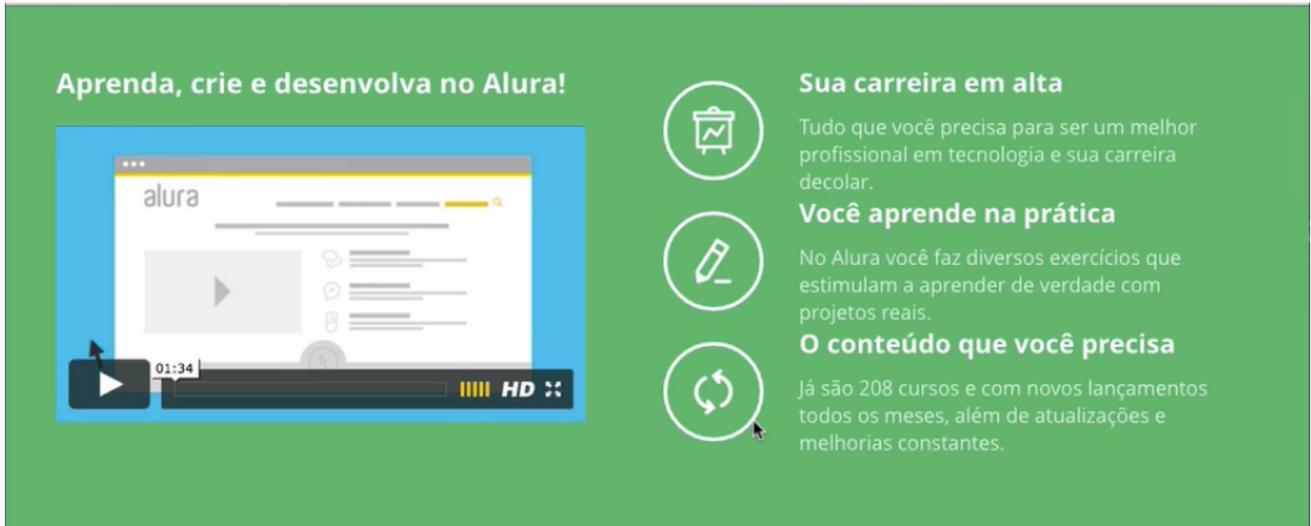
O que vamos fazer agora? Voltamos ao terminal e no arquivo `home-diferencias.css` vamos diferenciar os nossos ícones:

```

.icone-creca:before {
  background-position: center;
}
.icone-carreira:before {
  background-position: top;
}
.icone-impacto:before {
  background-position: bottom;
}

```

Então, diferenciamos as posições de cada ícone. Vamos atualizar e reparar como está nosso site:



Temos, finalmente, os três ícones que queríamos. Fizemos esse truque de juntar as imagens em um único arquivo e via *css* posicionamos para exibir apenas os ícones da vez, dando a impressão de que temos três arquivos separados, quando na verdade temos apenas um. Essa técnica que estamos utilizando é via *css* e se chama *sprite*. É muito interessante para economizarmos *requests*.

Bom, antes de encerrar vamos fazer uma discussão rápida. Viemos na pasta "site" original, editamos o *css* e geramos a imagem na pasta "site", ou seja, estamos manipulando tudo direto na pasta "site", a original, ao em vez de fazermos isso na pasta "dist", de produção.

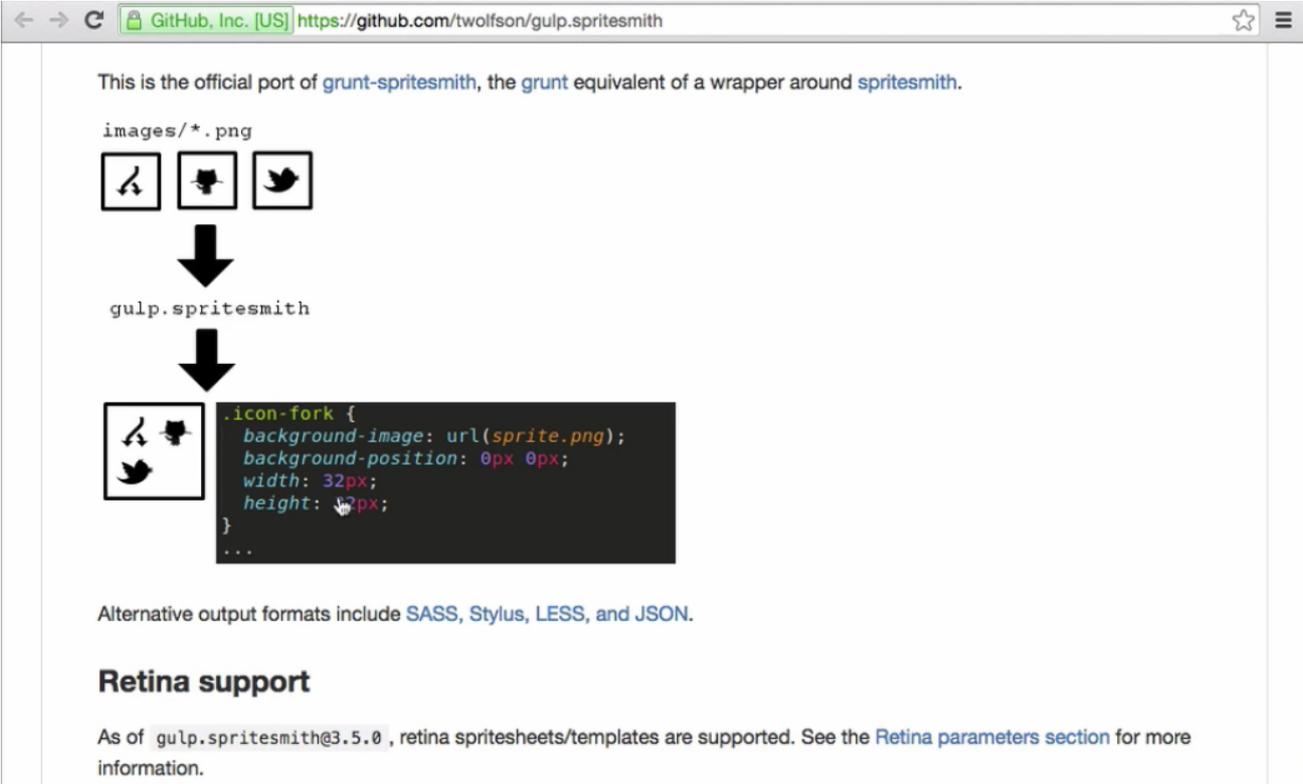
Aqui, existe muita controvérsia, há pessoas que acreditam que deveríamos fazer o site original com os arquivos separados e no site final é que gerariamos a *sprite*. Ou seja, deveríamos ter algum tipo de automação para gerar a *sprite*.

Tem pessoas que preferem manipular as *sprites* no arquivo original e fazem isso porque envolve uma série de *css* que podem ser um pouco complicados quando são gerados. É uma preferência escrever o *css* e saber que ele estará funcionando. Não quer dizer que esta imagem será desenhada no *photoshop* a mão, pois, foi apresentado, anteriormente, como gerar essa image. Esse método é um gosto pessoal, tem pessoas que preferem gerar o *css* final e para isso será necessário utilizar algumas ferramentas, como *pluggins* do *brunch*, para o *gulp*.

Por exemplo, o *Spritesmith* que está no link

(<https://github.com/Ensignten/spritesmith>)<https://github.com/Ensignten/spritesmith>

(<https://github.com/Ensignten/spritesmith>) e ele gera uma série de opções. Vejamos a página:



This is the official port of `grunt.spritesmith`, the `grunt` equivalent of a wrapper around `spritesmith`.

images/*.png



↓

gulp.spritesmith

↓



```
.icon-fork {  
  background-image: url(sprite.png);  
  background-position: 0px 0px;  
  width: 32px;  
  height: 32px;  
}
```

Alternative output formats include `SASS`, `Stylus`, `LESS`, and `JSON`.

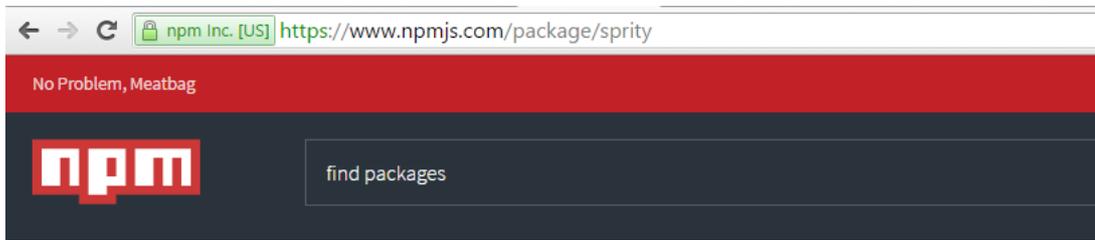
Retina support

As of `gulp.spritesmith@3.5.0`, retina spritesheets/templates are supported. See the [Retina parameters section](#) for more information.

Tem que cuidar para que o `css` gerado não entre em conflito com os demais e que ele seja inserido no local correto. As vezes, não é lá tão prático assim.

Outro *pluggin* é o *Sprity* que está no link (<http://www.npmjs.com/package/sprity>) (<https://www.npmjs.com/package/sprity>) que é um módulo do *npm* que usamos com *gulp* e podemos utilizar também na linha de comando e etc. Vamos observar como funciona isso. Na linha de comando basta utilizar o comando `create` e informamos onde queremos criar as imagens e quais as imagens que queremos usar.

Essa é a página dessa ferramenta:



★ sprity public

A image sprite generator

npm package 1.0.8 build passing coverage 96% code climate 4.0 dependencies insecure

- | A modular image sprite generator.
- | Generates sprites and proper style files out of a directory of images.
- | Supports retina sprites.
- | Can inline base64 encoded sprites.
- | Supports different output formats
- | Supports different image engines

Install

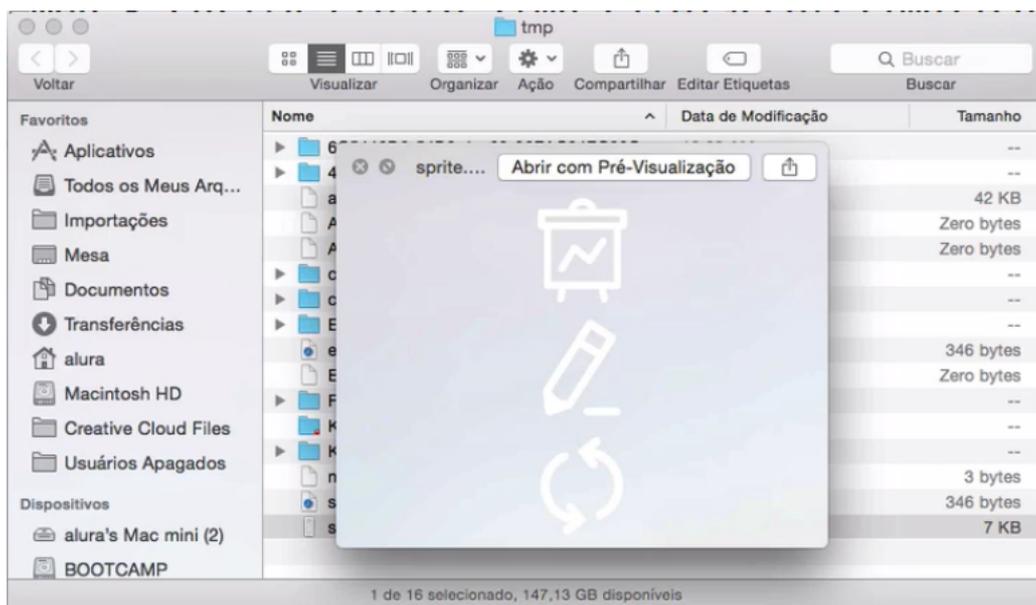
Install with **npm**

```
npm install sprity --save
```

E para usar esse comando na linha de comando basta digitar:

```
sprity create /tmp site/assets/img/icon-*.png .
```

E ele gera praticamente a mesma imagem que tínhamos antes:



Podemos passar também um `-s` que ele é capaz de gerar todos os `css` que queremos com as posições certas, para fazermos o *position*. Mas, se repararmos esse `css` parece, ainda, um pouco estranho, cheios de *icons*:

```

performance-web — bash — 79x20
bash
.icon {
  background-image: url('../images/sprite.png');
}

.icon-icon-diferencial-1 {
  background-position: -0px -0px;
  width: 88px;
  height: 88px;
}

.icon-icon-diferencial-2 {
  background-position: -0px -88px;
  width: 88px;
  height: 88px;
}

.icon-icon-diferencial-3 {
  background-position: -0px -176px;
  width: 88px;
  height: 88px;
}
performance-web $ █

```

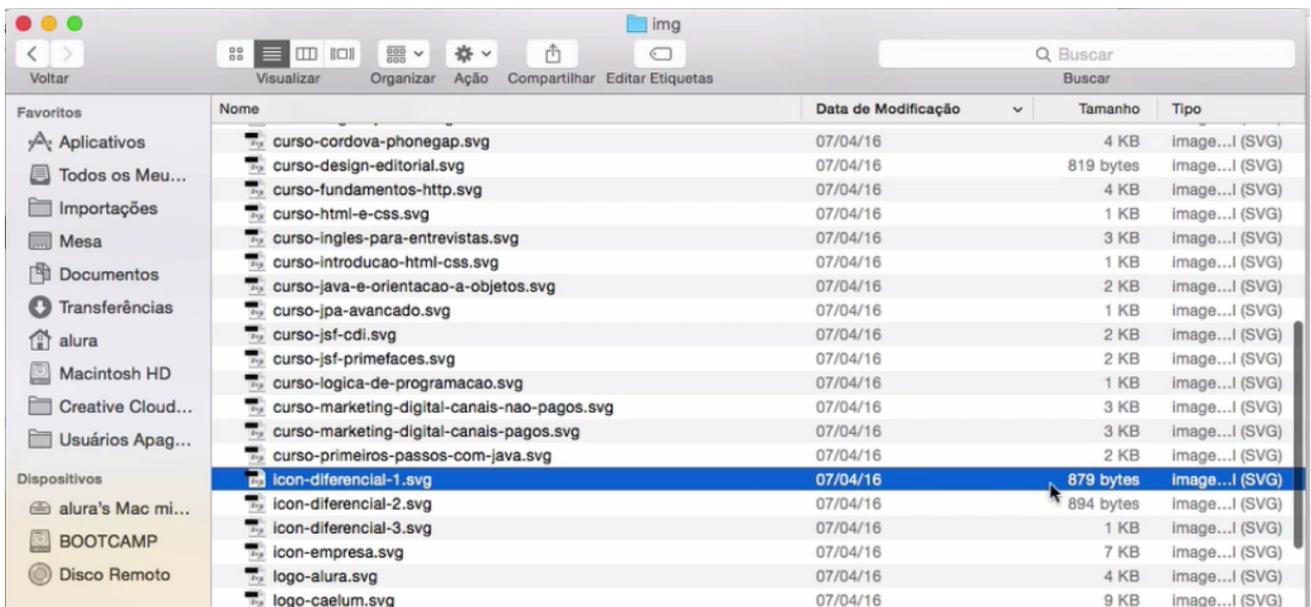
Bom, este é um código gerado que pode facilitar o trabalho.

Bom, existem esses dois meios de gerar as imagens. Uma mais manual, onde primeiro a imagem é gerada e depois o `css` é configurado corretamente. E a segunda que foi apresentada utilizando essas ferramentas.

Nesse nosso caso é preferível a maneira manual de fazer essa tarefa.

É claro que em casos de imagens muito grandes como foi o caso da imagem do *google*, essas ferramentas podem, efetivamente, ajudar.

Mostramos anteriormente como lidar com imagens e ícones utilizando o mecanismo de *sprite*. Em nosso caso temos uma maior parte das imagens em *svg*. Se repararmos no projeto, todos os ícones, praticamente, são em *svg*:



O *svg* é um formato que fica melhor em *mobile*, retina e etc e ele também é um formato menor. Então, devido a essas diversas vantagens, existe uma preferência por esse tipo de formato.

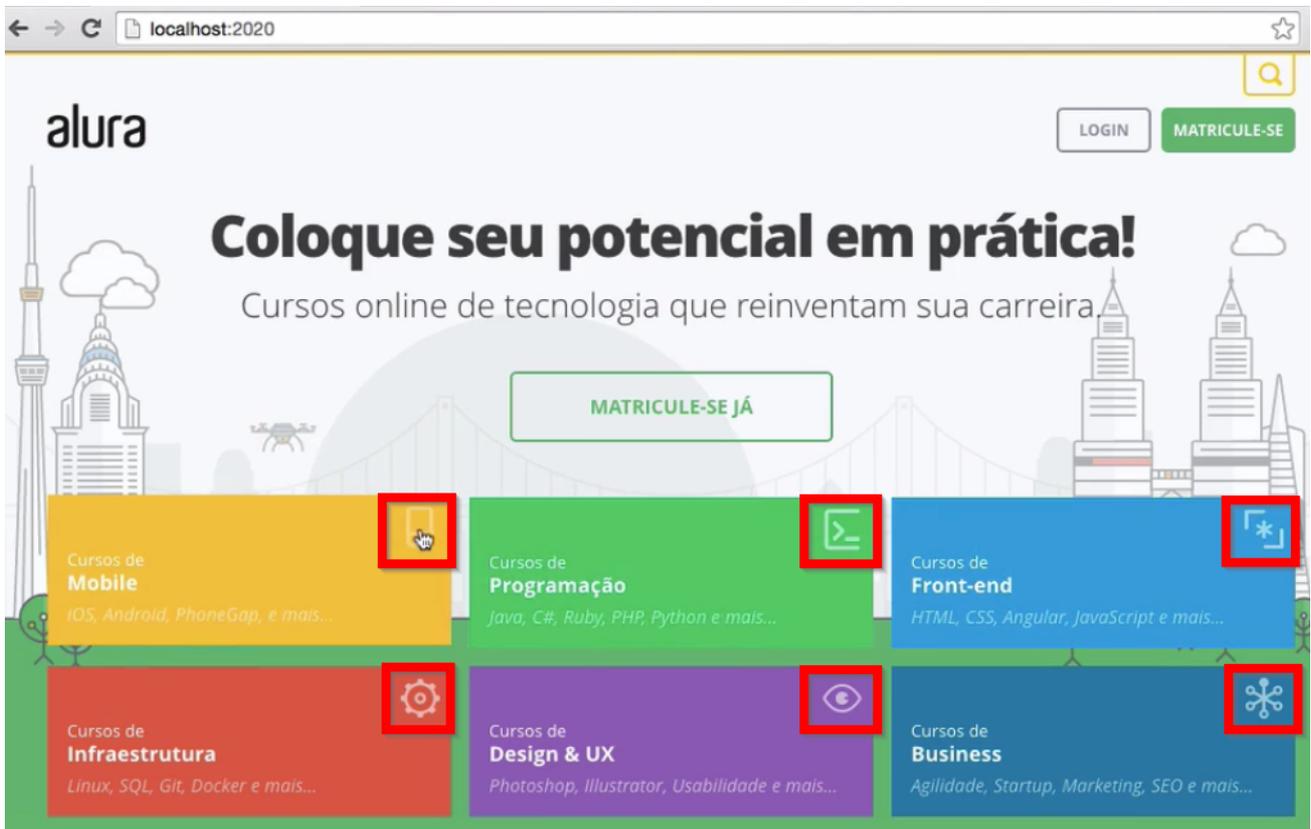
Mas, teremos o mesmo problema que antes, isto é, uma grande quantidade de arquivos. Poderíamos resolver esse problema agrupando as imagens em um mesmo arquivo e ajustando sua colocação e etc, mas existem outras maneiras de lidar com essa situação. Podemos lidar com o problema utilizando a ideia de *sprite*, só que diferente.

Quando fazemos o *sprite* na mão temos que reposicionar o *css* e isso pode ser um tanto trabalhoso. Com o *svg* temos outras possibilidades. Antes de explorar isso vamos observar o funcionamento do *categoria-business.svg* na prática. Vamos abrir um *svg* e vamos observá-lo, repare que ele nada mais é do que um *xml*. Ele possui diversos comandos que mostram como esse *svg* vai ser desenhado.

```
categoria-business.svg x
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <svg width="27px" height="30px" viewBox="0 0 27 30" version="1.1"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.
  org/1999/xlink" xmlns:sketch="http://www.bohemiancoding.com/
  sketch/ns">
3   <!-- Generator: sketchtool 3.4.4 (395) - http://www.
  bohemiancoding.com/sketch -->
4   <title>business</title>
5   <desc>Created with sketchtool.</desc>
6   <defs></defs>
7   <g id="Page-1" stroke="none" stroke-width="1" fill="none"
  fill-rule="evenodd" sketch:type="MSPage">
8     <g id="Desktop-HD" sketch:type="MSArtboardGroup"
  transform="translate(-1148.000000, -499.000000)" fill="
  #FFFFFF">
```

Estamos olhando isso na pasta "site", aquela que não está otimizada. Se observarmos o mesmo arquivo *svg* na pasta "dist" veremos que ele estará mais otimizado, os ícones estão menores.

Vamos conferir no *html* como os *svg* estão sendo chamados. E vamos pegar a parte das categorias. Repare que cada uma das seis categorias do site possui um ícone que é *svg*. Os ícones estão enquadrados de vermelho:



E como carregamos isso?

Vamos reparar no ícone do *Mobile*. Ele tem *img*, uma classe qualquer, *source* e categoria. E esse padrão se repete em todos os seis ícones. Imagine que queremos criar um *request* para que nele estejam os seis ícones.

Vamos fazer isso "à mão". Vamos na pasta "img" e criaremos um novo arquivo que será "categorias" e para visualizar que não é tão difícil fazer isso.

Vamos pegar a versão já minimizadas das categorias que fica mais fácil de vermos. Temos um ícone "categoria-mobile" e queremos colocar ele dentro do *svg*. Primeiro, criaremos um novo arquivo *svg*. Dentro dele, teremos uma tag chamada *defs* de *definition* e dentro dele teremos o que chamamos de símbolos que são os ícones. Dentro do *defs* colaremos os *svg* do *mobile*.

Vamos quebrar com espaços esse código para podermos entender o que temos nele:

```

1 <svg >
2 <defs>
3
4 <svg width="22" height="33" viewBox="0 0 22 33" xmlns="http://www
.w3.org/2000/svg">
5 <title>mobile</title><g id="Page-1" fill="none" fill-rule="
evenodd"><g id="Desktop-HD" fill-opacity=".6" fill="#FFF"><g
id="Page-1"><g id="Group-7"><path d="M17.612
29.584H3.604c-.384 0-.662-.304-.662-.761v-5.51h15.332v5.51c0
.457-.278.76-.662.76zM3.604 2.942h14.008c.384 0 .662.29.662.6
54v17.396H2.942V3.596c0-.364.278-.654.662-.654zm17.613.697A3.
64 3.64 0 0 0 17.577 0H3.64A3.64 3.64 0 0 0 0
3.64v25.242a3.64 3.64 0 0 0 3.64 3.64h13.938a3.64 3.64 0 0 0
3.64-3.64V3.64zM10.837 25h-.175C9.744 25 9 25.783 9
26.75s.744 1.75 1.662 1.75h.175c.919 0 1.663-.783
1.663-1.75S11.756 25 10.837 25" id="mobile"/></g></g></g></g>
6 </svg>
7
8 </defs>
9 </svg>

```

Vamos trocar a *tag* *svg* por uma chamada *symbol*. A ideia é que dentro do *defs* tenhamos vários *symbol*. Como acabamos de alterar o nome, vamos apagar o *xmlns* dele, pois não é mais necessário. E movemos o *xmlns* para o *svg* restante, que é a primeira *tag* do nosso arquivo.

Vamos, ainda, acrescentar depois da *tag* *svg* *width="0" height="0"* com isso estamos dizendo que não queremos que a imagem seja exibida e sim que sejam mostrados os símbolos individualmente.

```

1 <svg width="0" height="0" xmlns="http://www.w3.org/2000/svg">
2 <defs>
3
4 <symbol width="22" height="33" viewBox="0 0 22 33" >
5 <title>mobile</title><g id="Page-1" fill="none" fill-rule="
evenodd"><g id="Desktop-HD" fill-opacity=".6" fill="#FFF"><g
id="Page-1"><g id="Group-7"><path d="M17.612
29.584H3.604c-.384 0-.662-.304-.662-.761v-5.51h15.332v5.51c0
.457-.278.76-.662.76zM3.604 2.942h14.008c.384 0 .662.29.662.6
54v17.396H2.942V3.596c0-.364.278-.654.662-.654zm17.613.697A3.
64 3.64 0 0 0 17.577 0H3.64A3.64 3.64 0 0 0 0
3.64v25.242a3.64 3.64 0 0 0 3.64 3.64h13.938a3.64 3.64 0 0 0
3.64-3.64V3.64zM10.837 25h-.175C9.744 25 9 25.783 9
26.75s.744 1.75 1.662 1.75h.175c.919 0 1.663-.783
1.663-1.75S11.756 25 10.837 25" id="mobile"/></g></g></g></g>
6 </symbol>
7
8 </defs>
9 </svg>

```

Repare, é o mesmo código, só trocamos para *symbol*. E podemos introduzir diversas coisas aqui.

A ideia é que coloquemos as seis categorias.

E como usamos isso?

Vamos no *html* e para usar um *svg* dessa maneira precisamos usar a própria *tag* *svg*. Usaremos a *use* que é uma *tag* que recebe um *href* onde podemos passar o nome do *svg* que possui todos os ícones.

```
<svg><use xlink:href="assets/img/categorias.svg"/></svg>
```

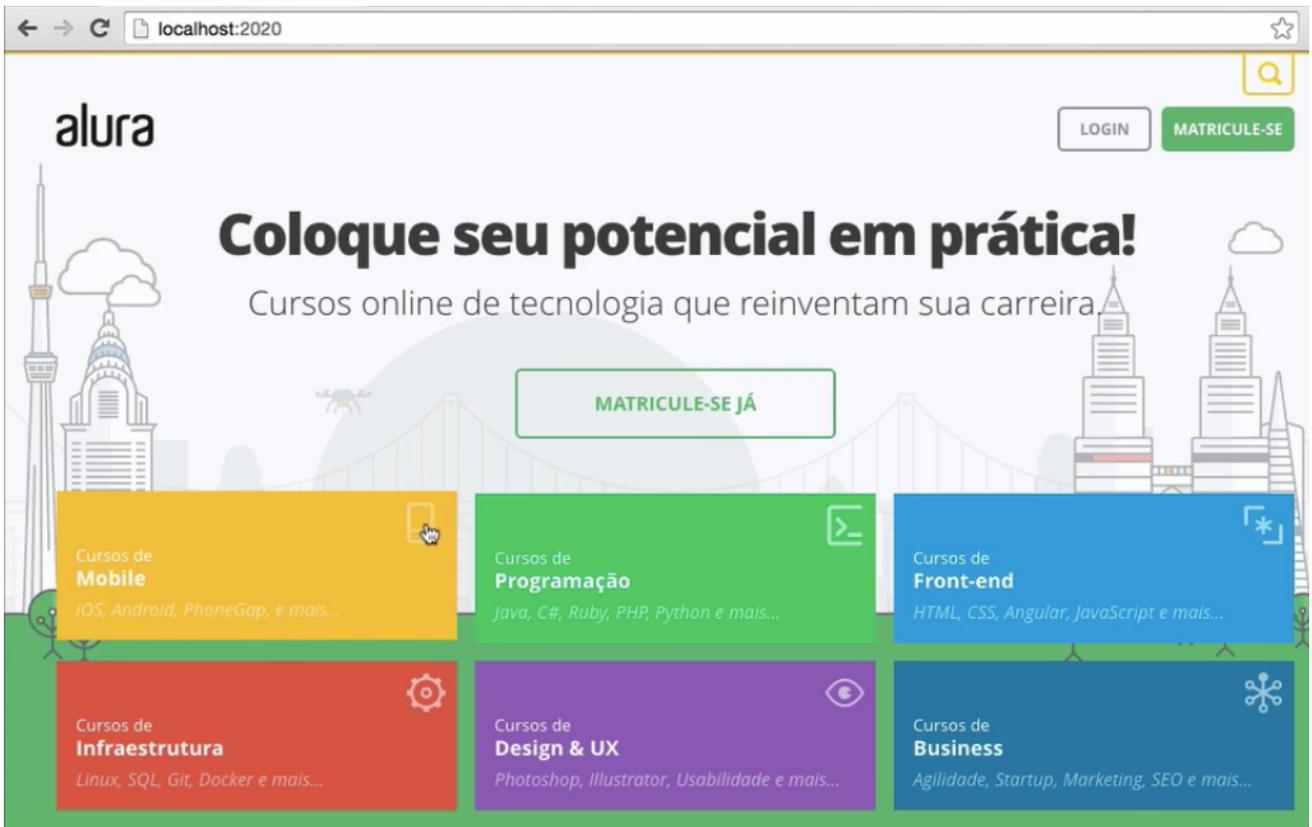
Note que na imagem abaixo temos o que fazíamos e o que estamos fazendo agora:

```
102 <!--
103 --><a href="/cursos-online-mobile" class="categoriaCard-item bg-categori
104 </svg>
107
108 <h3 class="categoriaCard-item-nome">
109 <div class="categoriaCard-item-nome-cursos">Cursos de</div>
110 Mobile
111 </h3>
112 <p class="categoriaCard-item-descricao">iOS, Android, PhoneGap, e ma
113 </a><!--
114 --><!--
```

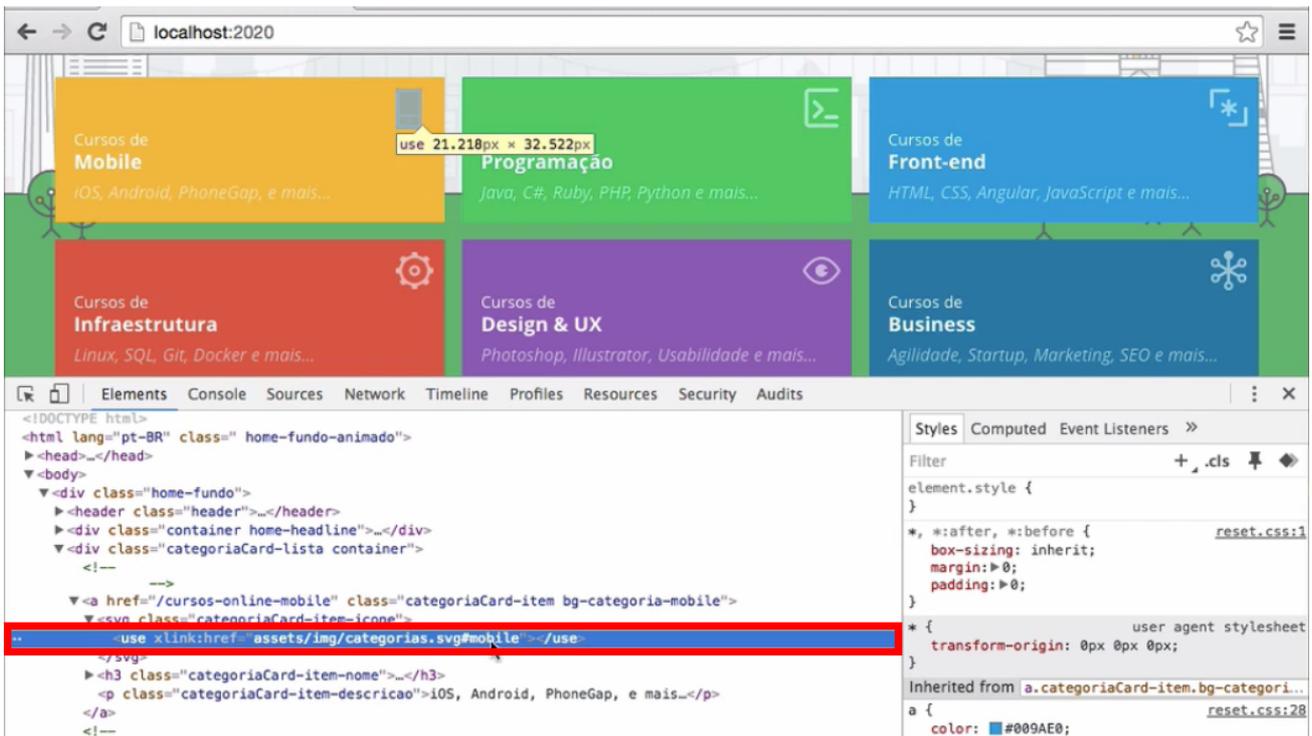
E agora, como ele sabe que ícone carregar? A ideia é que no *categoria* *svg* cada um dos símbolos que definimos pode ter um *id* que os diferencie, então, nomearemos o primeiro com o *id="mobile"*, *id="front-end"* e podemos continuar nomeando. E nesse momento, podemos ainda, no *"index.html"* adicionar no final do *use*, *#mobile*. Com isso estamos dizendo que queremos que isso seja carregado e lá dentro deve ser buscado o símbolo que contém *#mobile*.

```
101 <div class="categoriaCard-lista container">
102
103 <a href="/cursos-online-mobile" class="categoriaCard-item bg-categoria-mobile">
104 <img class="categoriaCard-item-icone" <use xlink:href="assets/img/categorias.svg #mobile"/></
105
106 <div class="categoriaCard-item-nome">
107 <div class="categoriaCard-item-nome-cursos">Cursos de</div>
108
109
110 <div class="categoriaCard-item-descricao">iOS, Android, PhoneGap, e mais...</p>
111
112
```

Vamos ver se funciona? Vamos atualizar nosso navegador e, aparentemente, constatamos que tudo está em ordem:



Podemos dar um *inspector* e confirmar que tudo está funcionando nos conformes:



E a mesma coisa se repete para os próximos itens, podemos fazer o mesmo com o front-end adicionando #front-end .

Repare, que fazendo isso estamos economizando *requests*. A criação desse arquivo é muito simples, a ideia é juntar os arquivos e usar a tag *symbol*.

E o que fazemos? Mantemos os ícones separados e escrevemos na mão?

Podemos fazer uma geração automática utilizando algumas ferramentas. Podemos procurar no *google*, o *svg sprite*.

Acessaremos o link (<https://github.com/jkphl/svg-sprite>)<https://github.com/jkphl/svg-sprite>

(<https://github.com/jkphl/svg-sprite>). Ele mostra diversas configuração e opções, podemos fazer isso automaticamente, temos, inclusive, um *pluggin* para o *gulp*. Ou, podemos criar isso nos próprios editores que existem para *svg* que suportam exportar direto para a sintaxe do *symbol*. Por exemplo, podemos utilizar o *Inkscape* para desenhar ícones e também podemos utilizá-lo para visualmente rotulá-lo de símbolo, assim, ele já gera a sintaxe automaticamente, sem a dificuldade de fazer isso a mão.

Então, temos essas opções: (1) usamos um gerador qualquer; (2) fazemos isso a mão (3) usamos um editor visual;