Bootstrap

do jeito certo

Produzido em parceria com a Agência Zíriga

AGÊNCIA FULL DIGITAL

www.ziriga.com.br

O Mestre Linguiça

Keven Jesus

{

Paulistano meio nordestino, fundador da maior comunidade de Bootstrap do Brasil, desenvolvedor Front-end especialista em arquitetura de interfaces e apaixonado por tecnologias Javascript como Angular, React e outros.

}

Índice

{INTRODUÇÃO}

Criado por dois desenvolvedores que trabalharam no Twitter e cuja fascinação por CSS, **mdo** e **fat** fez com que fossem ainda mais adiante, o Bootstrap surgiu como uma maneira prática de melhorar a produção de layouts responsivos para a web. Antes pensado e utilizado internamente por seus idealizadores, o Bootstrap, por ter sua codificação em ***open source***, foi logo descoberto e fez com que o programa se tornasse extremamente famoso entre todos os desenvolvedores.

Como você já deve saber, o Bootstrap se trata de um ***framework*** voltado para HTML, CSS e JS que permite que projetos mobile e responsivos para a web sejam criados rapidamente. Adequar os avanços da programação para web com as constantes demandas do mercado por layouts mais modernos e um design mais intuitivo tem sido uma tarefa árdua para muitos desenvolvedores. Contudo, ao oferecer excelentes vantagens como sua fácil customização, as atualizações recorrentes e componentes intuitivos que facilitam a experiência do usuário, já deu pra entender o porquê da popularidade do Bootstrap, não é?

E para tornar o Bootstrap ainda mais prático, vou lhe ajudar a descobrir tudo que ele pode lhe proporcionar! Unindo conceitos e ferramentas como o Gulp e o Sass à sua dinâmica, é possível automatizar o número de tarefas que irão se apresentar no processo de construção de uma página web responsiva e assim reduzir o tempo que será gasto. Com isso, você também poderá dar uma maior atenção para outros projetos que está realizando. Todos saem ganhando!

No eBook que você irá conferir a seguir, irei lhe apresentar todas essas novas formas de organização e otimização da sua experiência com o Bootstrap, para tornar o que já era fácil em quase um mamão com açúcar. Ficou curioso? Conheça mais sobre o material abaixo e também traga a simplicidade do Bootstrap para o seu ambiente de trabalho.

NODE PACKAGE MANAGER

Antes de iniciarmos quaisquer tarefas com o Gulp, é preciso falar um pouco sobre a NPM e o porquê que devemos utilizá-la.
A NPM se trata da sigla utilizada para se referir ao Node Package Manager (Gerenciador de Pacotes do Node). Entre suas descrições, deve-se destacar duas que nos ajudam a compreender melhor toda sua extensão. De início, a NPM serve como um repositório online para que sejam publicados projetos de código aberto para o Node. Em complemento a isso, ela também atua como um utilitário de linha de comando que interage com este repositório, ajudando na instalação de pacotes e no gerenciamento de dependências.

Por se tratar de um programa de código aberto, a NPM conta com diversos pacotes com novas aplicações que são incluídas diariamente e que são facilmente encontradas em seus portais de busca. Basta encontrar aquilo que deseja instalar e digitar uma única linha de comando para dar início. Para exemplificar, o que precisamos fazer é criar um arquivo manifest de configuração e responder algumas perguntas como nome do projeto, descrição e tudo mais, sempre rodando no terminal.

```
$ npm init
```

Após responder todas as perguntas do NPM, como nome do projeto, descrição, palavras chaves, etc, seu arquivo de configuração (manifest) será gerado com nome de package.json.

```
{
  “name”: “projeto”,
  “version”: “1.0.0”,
  “description”: “”,
  “main”: “index.js”,
  “scripts”: {
    “test”: “echo \”Error: no test specified\” && exit 1”
  },
  “author”: “Keven Jesus”,
  “license”: “ISC”
}
```

{BOWER}

GERENCIADOR DE PACOTES

Não, não é o herói daquela série de ação “24 Horas”. Nosso Bower é uma ferramenta criada para gerenciar componentes para desenvolvimento ***front-end***. Independentemente da origem do seu back-end, sendo Java, Ruby, Python, Node ou PHP, ele pode ser adicionado em qualquer projeto que esteja executando.
Para fazer a instalação do Bower, você irá precisar ter o Node junto ao seu NPM também instalado em seu computador. Caso o contrário, a operação não ocorrerá normalmente.
Vá ao seu terminal e utilize a flag **-g bower**. Dessa maneira, ele será executado globalmente em toda sua máquina

```
$ npm install -g bower
```

Para incorporar o Bower em um projeto web simples, vamos imaginar a seguinte estrutura:

|certificacao-bootstrap
|--source
|----sass
|------scss
|--------style.scss
|--index.html

Logo, para adicionar o Bower, você deve seguir até a pasta do projeto e digitar no terminal **bower init**. Depois, o Bower vai dar início a um wizard para gerar o arquivo **bower.json**, que também irá pedir para você completar algumas informações como nome do projeto, sua descrição, palavras chave, etc.
Quando toda a operação estiver finalizada, surgirá um novo arquivo bower.json similar ao que você vê abaixo:

```
{
  name: ‘projeto’,
  description: ‘ebook com bower’,
  main: ‘’,
  license: ‘MIT’,
  homepage: ‘’,
  ignore: [
    ‘**/.*’,
    ‘node_modules’,
    ‘bower_components’,
    ‘test’,
    ‘tests’
  ]
}
```

No eBook, vamos utilizar o Bootstrap como dependência, como é costumeiro encontrar em projetos front-end. Para entender isso e adicionar a dependência em seu projeto, você irá digitar no terminal.

```
$ bower install bootstrap --save
```

Assim, toda vez que você instalar um novo pacote, ele irá salvar no arquivo **bower.json**.

```
{
  “name”: “projeto”,
  “description”: “ebook com bower”,
  “main”: “”,
  “license”: “MIT”,
  “homepage”: “”,
  “ignore”: [
    “**/.*”,
    “node_modules”,
    “bower_components”,
    “test”,
    “tests”
  ],
  “dependencies”: {
    “bootstrap”: “^3.3.7”
  }
}
```

O interessante do Bower é que ele trabalha com algo chamado dependência transitiva. Isso significa que se um pacote depende de algum terceiro, ele será baixado juntamente no download.

Se você ainda deseja inserir novos componentes em seu projeto e não sabe como adicioná-los, há a opção de busca através do **bower search**, Exemplo: caso queira adicionar o Bootstrap ao seu projeto e não sabe como localizar o pacote correto, digite **bower search bootstrap** e voilá! Ainda que seja um tanto quanto confuso, é uma maneira de facilitar sua vida.

{GULP}

O AUTOMATIZADOR DE TAREFAS

Ao criar uma página para web ou realizar qualquer outro tipo de atividade em seu trabalho enquanto programador e/ou desenvolvedor, é comum que você acabe acumulando diversas tarefas para realizar. E mais comum do que isso, são casos onde essas tarefas, por mais obrigatórias e necessárias que sejam, acabam se tornando verdadeiras “pedras” no sapato. Sua verdadeira preocupação nesse momento deve ser em automatizar algumas destas tarefas para manter o foco em seu resultado.

Essa mecanização é fundamental, pois na correria de nosso dia-a-dia enquanto programadores e/ou desenvolvedores, o tempo para executar essas tarefas tem se tornado cada vez mais escasso. Ao realizar anotações sobre tudo que devemos realizar, é possível que se esqueça de algo que poderá ocasionar maiores problemas posteriormente. É importante estabelecer determinados padrões nesse sentido para que sua atenção seja voltada uma única vez para essas tarefas primárias.

Por isso, “bora” falar mais sobre o Gulp? Se você já o conhece, tenho certeza que sabe do grande auxílio que ele vem prestando para muitos profissionais. Caso você ainda não o conhece, estou sentindo que vai ser amor à primeira vista!

Quem é o Gulp?

O Gulp é um plugin do Node que funciona como um automatizador de tarefas, com sintaxe JavaScript. É importante ter o Node instalado em sua máquina. Acima de qualquer definição, o Gulp será um de seus melhores amigos a partir de agora. Sua utilização é muito simples e irá tornar tudo mais gerenciável. Ele também mantém um alto padrão de performance, sendo constantemente melhorado pela comunidade.

Ao entrar em sua página (gulpjs.com), você poderá conhecer mais sobre todos os plugins existentes que irão lhe auxiliar, tendo cada um sua própria funcionalidade, como o **gulp-sass** e o **gulp-minify-css**, por exemplo. Para instalar e executar o Gulp em sua máquina, veja os tópicos a seguir.

Instalando plugins

Primeiramente, você precisa ir até o terminal, tendo como base que seu sistema operacional seja tanto Mac quanto Linux. Se for o seu caso, o **sudo** pode se tornar necessário caso sua máquina esteja com alguma configuração diferente de permissão. Nela, você irá instalar o Gulp através do NPM (Node Packager Manager) que citei anteriormente, para que o comando Gulp seja reconhecido globalmente através do terminal. Ao digitar as iniciais para instalar o plugin do Node (Gulp), é necessário utilizar a flag **-g (global)** junto ao NPM:

```
$ npm install -g gulp
```

Após essa etapa ter sido concluída, é necessário entender como instalar também outras ***features*** do Gulp, como o **gulp-sass**, por exemplo. Você vai ouvir muito sobre ele ainda, mas vamos por partes, como diria Jack, O Estripador. Tomando o **gulp-sass** como plugin base, retorne ao terminal.

```
$ npm install --save-dev gulp-sass
```

Neste caso, a diferença é que a instalação irá ser adicionado diretamente para um repositório chamado **node_modules**, e criada uma subpasta do plugin chamada **gulp-sass**. Para ter certeza que o plugin foi instalado com sucesso, é só você ir até a pasta **node_modules**. Ao expandir essa área, basta você encontra a subpasta chamada **gulp-sass**, além de também encontrar a pasta primária do próprio Gulp. Se preferir, você também pode ir até o arquivo manifest **package.json** e conferir que o **gulp-sass** está devidamente adicionado.

```
{
  “name”: “projeto”,
  “version”: “1.0.0”,
  “description”: “”,
  “main”: “index.js”,
  “scripts”: {
    “test”: “echo \”Error: no test specified\” && exit 1”
  },
  “author”: “Keven Jesus”,
  “license”: “ISC”,
  “devDependencies”: {
    “gulp-sass”: “^2.3.2”
  }
}
```

O **gulp-sass** será o plugin responsável por transcrever nosso código escrito em Sass e gerar o CSS para o navegador fazer a leitura.

Registrando tarefas

Agora que você já conta com o Gulp, chegou a hora de dar início aos processos! Será necessário que você crie um arquivo de configuração na pasta raiz, que irá registrar as tarefas que deseja executar em sua automatização. Sempre lembrando que o próprio plugin foi desenvolvido em JavaScript, crie seu arquivo como **gulpfile.js** e crie uma variável que será responsável por armazenar o Gulp.

```
var gulp = require("gulp");
```

O Node segue a CommonJS, uma especificação de ecossistemas para o JavaScript, e a função embutida **require** é a maneira mais fácil de incluir módulos existentes em arquivos separados. O funcionamento básico do require é que ele lê o arquivo JavaScript, interpreta o script e em seguida retorna o conteúdo do objeto **exports**.
Após isso, você irá criar uma nova variável, repetindo o mesmo padrão de comandos, porém invocando o plugin **gulp-sass**.

```
var sass = require("gulp-sass");
```

Os conhecidos parênteses surgirão e, entre eles, você irá determinar dois parâmetros, sendo o primeiro o nome da tarefa em string e o segundo, uma função anônima para que a mesma seja executada. Para exemplificar, imagine que o primeiro parâmetro seja **compilar-css** e o segundo seja **function(){}**, sendo ambos separados por vírgula.

Com tudo feito, é hora que criar sua tarefa. Você terá que tomar dois passos agora. No mesmo arquivo **gulpfile.js**.

```
gulp.task("compilar-css",  function(){

});
```

Dessa forma, suponha que você tem um arquivo **style.scss**, precise transformá-lo em CSS nativo e ainda repassá-lo para outro diretório. Você deve ter ficado cansado só de pensar, não é? Logo , basta você criá-la uma vez e automatizá-la para ficar tranquilo.

Na pasta inicial do projeto, crie um novo diretório chamado source e crie uma nova pasta chamada ***scss*** dentro do mesmo. Nessa nova pasta **scss**, crie um arquivo, que em nosso caso se chamará **style.scss**, onde você escreverá um CSS aleatório, por exemplo.

```
body {
    background-color: #333;
}
```

Suponha que você precise encontrar esse novo arquivo a partir do Gulp e ainda não saiba como começar. Você precisa percorrer e retornar esse arquivo para o plugin **gulp-sass**, que está armazenado na variável sass criada anteriormente.

```
gulp.task("compilar-css",  function(){
  return gulp.src("./source/scss/style.scss")
});
```

Há duas formas existentes para que se percorra os caminhos pelo Gulp. A mais conhecida é fazer sua busca pelo seu arquivo, ao digitar **./**, que simbolizam a entrada na raiz do diretório padrão. Em seguida, também será repassado o caminho do arquivo (**./source/scss/style.scss**), fazendo referência ao último arquivo. A segunda forma é passar um array em JavaScript, onde você poderá procurar por diversos arquivos por seus nomes. Isso pode ser feito para inúmeros arquivos que estejam presentes em outros diretórios.

Caso você tenha inúmeros arquivos, não saberia encontrá-los e também não deseja passar um array de todos eles, o * (asterisco) representa todos esses arquivos junto com a extensão final **scss**. Assim, você consegue encontrar todos os arquivos que pertençam à ela.

```
gulp.task("compilar-css",  function(){
  return gulp.src("./source/scss/.scss")
});
```

Continuando, na linha abaixo, vamos acrescentar o .pipe. Ele se trata uma convenção do Node para leitura. É aqui onde a transformação do Sass para CSS irá acontecer. Dentro da linha, invoque a variável **sass** (que representa o **gulp-sass**) como função,

```
gulp.task("compilar-css",  function(){
  return gulp.src("./source/scss/*.scss")
          .pipe(sass())
});
```

Em uma nova linha, você deve rodar o pipe novamente e informar para onde será enviado o arquivo transcrito, utilizando o **gulp.dest** e passando o diretório.

```
gulp.task("compilar-css",  function(){
  return gulp.src("./source/scss/*.scss")
         .pipe(sass())
         .pipe(gulp.dest("./dist/css"))
});
```

Um fato interessante sobre o Gulp é que quando o diretório em questão não existe para executar a função em questão, ele o cria. Mas se ele existir previamente, ele só irá sobrescrever sob o pré-existente. Em resumo, o que está acontecendo é o retorno de um arquivo que será criado no novo diretório. A primeira linha, onde inserimos o comando return, você estará procurando pelo(s) arquivo(s). A segunda linha com o pipe pega o arquivo Sass e o transcreve para CSS. E na terceira e última linha do pipe, você o recebe agora totalmente compilado e direcionado. Muito simples, né não?
Para testar e comprovar que tudo esteja funcionando adequadamente, abra seu terminal e digite para testar a tarefa criada.

```
$ gulp compilar-css
```

Ao dar enter, perceba sua atividade foi concluída quase instantaneamente e que foram criadas duas novas pastas (**dist e css**) e um novo arquivo (**style.css**) no diretório css. E voilá! Sua primeira tarefa foi totalmente automatizada com sucesso.
Toca aqui!

Watch: monitorando arquivos

Um outro recurso que vale a pena comentar é o Watch. O Watch é uma feature que permite monitorar seus arquivos. Mas você pode (ou provavelmente deve) estar se perguntando "mas como assim monitorar os arquivos, Keven?!". Pois bem, imagine que você tem o comando para compilar em formato css, mas toda vez que uma mudança terá que ser feita, também passará pelo mesmo processo de abrir seu terminal, executar um novo comando, fazer isso manualmente para cada ocasião… Admita, ninguém está afim de passar a tarde fazendo isso.

Para resolver essa questão, você irá criar uma nova task do Gulp chamada "escuta-me", que ficará encarregada de monitorar todas as atividades passadas para ela. Então, repita o processo de criar uma tarefa, passando nome e a função anônima.

```
gulp.task("escuta-me",  function(){

});
```

Abaixo, digite o comando na linha abaixo com **gulp.watch**.

```
gulp.task("escuta-me",  function(){
   gulp.watch(" ",[]);
});
```

Teremos dois parâmetros para estabelecer: "quem" e "para onde". "Quem" se trata de qual arquivo estará sendo monitorado.
No caso, todos os arquivos. Dessa forma, é hora de repetir o padrão anterior **./source/scss/*.scss**, para que ele execute assim todos os arquivos no formato scss.

```
gulp.task("escuta-me",  function(){
   gulp.watch("./source/scss/*.scss",[]);
});
```

Para o parâmetro "para onde", você irá inserir um array de tarefas que podem ser executadas quando houver mudanças nos arquivos com extensão scss. Ambos os parâmetros são complementares ao outro, onde o arquivo foi "escutado" e a função será executada de acordo com o que foi programado.

```
gulp.task(“escuta-me”,  function(){
   gulp.watch(“./source/scss/*.scss”,[“compilar-css”]);
});
```

Salve seu progresso e vá até o terminal e rode sua tarefa **escuta-me** e pronto: ele está executando!

```
$ gulp escuta-me
```

Detalhe importante: a tarefa escuta-me está acontecendo normalmente após esse passo, mas não a compilar-css. Essa última parte só será executada quando você se modificar o arquivo **style.css** e salvar. Se você olhar novamente no terminal, verá que agora ela estará rodando, sem ter fechado nenhum outro processo, apenas se você pressionar modificar e salvar. Nesse caso, basta apenas repetir os comandos de cada em seu terminal para que fiquem operando em background.

{SASS}

POR QUE UTILIZÁ-LO COM O BOOTSTRAP

Conforme utilizei como exemplo em minha abordagem no tópico anterior para demonstrar a utilização do Gulp como um "facilitador" de tarefa, para que se compreenda toda a abrangência dessa mecanização, é muito importante que também se fale um pouco mais sobre o Sass. Para começar, do que se trata o Sass?

Tenho certeza que, um belo dia, em sua nada mole vida de desenvolvedor, você estava executando a mesma folha de estilo mais de 10, 15 ou até 20 vezes, seja ela para qualquer tipo de especificação. Copiando e colando, copiando e colando…

Fala sério, né? Deveria haver uma maneira mais prática e direta para que essa programação seja feita e lhe poupar alguns bons minutos. É nesse momento em que o Sass entra em ação para nos ajudar. Ele funciona como uma espécie de extensão em linguagem CSS, algo como um pré-processador, onde você irá escrever um css, mas com super poderes, sendo extremamente parecida com a linguagem tradicional. Algo que eu gosto de chamar de "CSS inteligente".

"Mas por que 'CSS inteligente', Keven?" é a pergunta da vez. Isso pode ser afirmado devido aos grandes benefícios que ele lhe oferece, como:

Variáveis

Enquanto desenvolvedor, tenho certeza absoluta que tenha aprendido e visto de perto o poder de uma variável. Mas, caso isso seja um conceito um tanto quanto estranho para você,
ficarei feliz em lhe explicar. Imagine um guarda-roupa e suas
gavetas. A função primordial de toda gaveta é de armazenar qualquer tipo de coisa, sejam suas roupas, seus calçados, seus livros, etc. As variáveis funcionam exatamente da mesma maneira: elas servem para armazenar valores.

Modularização

O Sass é um excelente auxiliar para projetos modularizados. Em suas configurações, o programa possui um comando **import** que se diferencia muito do tradicional import do CSS.

Reuso

Bem, se você tem funcionalidades de uma classe que tenha uma importância grande e que será utilizada novamente em seu projeto, duplicando seus códigos, é comum ouvirmos relatos de problemas que isso possa gerar. Com a reutilização que o Sass pro proporciona, você poderá trazer esses valores de volta de uma maneira simples com ***mixins***, ***placeholders*** e ***funções***.

Custo de Manutenção

Graças aos fatores anteriores, isso também implica em um baixo custo de manutenção com Sass. Assim, todos os processos que tomariam muito tempo para serem resolvidos poderão ter soluções em questão de minutos. Tudo isso contribui para que se enxergue o Sass não apenas como um método de auxílio imediato, mas como um verdadeiro investimento para projetos a médio e longo prazo.

Operações Matemáticas

Um dos grandes baratos do Sass é que ele apresentar operações matemáticas como adição, divisão, multiplicação e subtração por suportar esse tipo de programação, como também cálculos envolvendo dados percentuais, arredondamentos, etc. Tudo isso garante um leque de opções mais amplo em seu campo de trabalho.

Funções e Condicionais

Acima de tudo, é extremamente importante ressaltar que o Sass é composto por funções e condicionais. Você é capaz de fazer ifs no seu CSS. Pois é, meu caro. É isso mesmo que você está lendo. Todo esse poder na palma da sua mão. Mas calma, pois vem muito mais a seguir!

Bootstrap

A versão 4 do Bootstrap foi totalmente reescrita em Sass. Então, nem preciso comentar muito sobre a importância e o envolvimento do Sass com o framework, não é mesmo?

Variáveis

Como um primeiro exemplo para explicar e destrinchar as formas de atuação do Sass, escolhi abordar uma em específico que acredito que seja seu maior diferencial perante o restante: as variáveis. Você vai entender um pouco mais sobre como o isolamento de dados ocorre com o uso das variáveis. Para começarmos, imagine o cenário onde você tem um componente card e ele corresponda com alguns valores, sendo que esses possam ser isolados em variáveis num ambiente com sass.

CSS TRADICIONAL

```
.card {
   padding: 20px;
   width: 100%;
}
```

SCSS (SASS)

```
 .card {
    padding: $card-padding;
    width: $card-width;
 }
```

Ao utilizar uma variável com o Sass, é preciso iniciar com o cifrão dólar. Em seguida, informar o nome da variável, sempre respeitando a regra de que não se pode iniciar com número, ter espaços ou caracteres especiais como @ (arroba), # (hashtag), ! (exclamação), etc. Após isso, foi informado o valor dessa variável, sendo nosso caso de 20px e 100% .

```
$card-padding: 20px;
$card-width: 100%;

.card {
   padding: $card-padding;
   width: $card-width;
}
```

Árvore de seletores e reutilização de classes

Após tantas expressões que formulei para lhe ajudar a entender melhor as funções do Sass, vamos falar um pouco mais sobre a "árvore de seletores". Como você já sabe, uma árvore possui diversos galhos que se espalham ao seu redor, partindo sempre de um ponto em comum, no qual seria o tronco. Pois bem. No CSS tradicional, normalmente encontramos uma situação onde temos uma classe header e para que incluamos um logo em conjunto, digitamos **.header** e **.logo**. Fazendo isso, queremos dizer que a classe **.logo** pertence à classe **.header** em questão, como um elemento filho.
No Sass, isso se resolve de uma maneira visualmente mais interessante e intuitiva. É quando voltamos para nossa árvore do começo. A disposição das configurações se assemelha aos galhos e nos oferece uma liberdade muito maior. Dê uma olhadinha aqui embaixo para entender melhor.

CSS TRADICIONAL

```
.header { background: #fff; }
.header .logo { background: #000; }
```

SCSS (SASS)

```
.header {
   background: #fff;
    .logo{
        background: #000;
    }
}
```

Primeiramente, antes de qualquer outro passo que seja apresentado a partir de agora, gostaria de ressaltar que **é extremamente importante que você tenha um conhecimento mínimo de CSS e HTML**. Digo isso porque o Sass trabalha com esse tipo de linguagem e não faz sentido estar aqui explicando algo que você não saberia como desenvolver por conta própria. Por isso, esse é um dos pré-requisitos que abordo em relação aos interessados em realizar meu curso, no qual vamos falar um pouco mais em breve.
Se deseja dominar essa ferramenta por completo, procure conhecer um pouco mais a respeito das funcionalidades do CSS, HTML e JavaScript. Você se sentirá melhor integrado com todo o conteúdo que estou explicando aqui e será o pontapé inicial para um novo mundo em sua carreira. Confie em mim.
Ok, você pôde ver que, em termos visuais, essa interface de seletores do Sass é uma opção muito melhor. Mas ela não para por aí...

Pseudo elementos

Os conhecidos pseudo-elements também fazem parte dessa árvore e podem ser acrescentados da seguinte maneira.

CSS TRADICIONAL

```
.header { background: #fff; }
.header:hover { background: #000; }
```

SCSS (SASS)

```
.header {
   background: #fff;
   &:hover{
     background: #000;
   }
}
```

Veja que surgiu um novo “carinha”, o famoso “e” comercial. Ele garante diferentes acessos aos elementos no Sass, com um deles podendo ser usado para pseudo-elementos.

{BOOTSTRAP}

SEMÂNTICA

É, meus amigos… Estamos chegamos ao nosso último tópico! A partir de agora, vamos abordar mais sobre o próprio Bootstrap e como podemos compreendê-lo ainda mais para otimizar sua atuação em prol do seu trabalho. Algo que gostaria de abordar logo de início é aquilo que seria a semântica do Bootstrap. Muito se fala de um grande "problema" do Bootstrap. Digo isso porque se trata de um fator que causa uma colisão de nomes, uma espécie de inconsistência na manutenção de seu projeto.

Uma forma de ilustrar e descobrir o que estou dizendo é criar um arquivo HTML e uma estrutura simples. Por exemplo, uma box de perfil.

```
<div class="about col-md-9 col-lg-6">
    <div class="about-image col-md-6">
        <img src="images/foto.jpg" alt="perfil">
    </div>
    <div class="about-bio col-md-6">
        <h1>Keven Jesus</h1>
        <p>Lorem ipsum dolor sit amet,
       consectetur adipisicing elit.</p>
    </div>
</div>
```

Já deu pra notar que sua programação começou a se tornar uma grande bola de neve, não é? Em um projeto em HTML, uma apresentação de seu processo se torna um tanto quanto ilegível. Mas espere! Foi para isso que rumamos juntos até aqui. Para esse momento. Esse tópico. Aquele que fará com que tudo que lhe foi apresentado ganhe muito mais sentido. Vem comigo!

Ao nos dirigirmos ao arquivo **style.scss**, localizado na pasta sass, vamos incluir o core do framework, utilizando o comando **import**, percorrendo até a pasta **components** e procurando o principal arquivo Sass do Bootstrap.

```
@import "../components/bootstrap/scss/bootstrap";
```

Ao nos dirigirmos ao arquivo **style.scss**, localizado na pasta sass, vamos incluir o core do framework, utilizando o comando **import**, percorrendo até a pasta **components** e procurando o principal arquivo Sass do Bootstrap.

```
@import "../components/bootstrap/scss/bootstrap";
```

Ao retornar para o **index.html**, copie e cole toda a box **.about** construída em nosso processo e deixe comentado. Isso servirá como base para termos uma noção espelhada das diferenças entre uma box e outra. Vá ao navegador e confirme que a estrutura não foi duplicada. Perfeito? Muito bem. Após isso, em sua segunda box, retire todos as cols do Bootstrap que incluímos antes, deixando apenas as classes iniciais (**about**, **about-image**).

"Mas Keven, como assim tirar as cols? Esse negócio todo não é sobre Bootstrap?"

Sim, você está certo e vai entender o que estou lhe propondo muito em breve. Primeiramente, você deverá alimentar este **.about** e seus subsequentes (**.about--image** e **.about-bio**) no **style.scss**, incluindo características visuais. Sendo feito, basta salvar e dar início às tarefas **compilar-css** e **escuta-me** em seu terminal. Ao atualizar seu navegador, seus boxes terão fundos preenchidos com elementos visuais que você determinou.

```
@import "../components/bootstrap/scss/bootstrap";

.about{
  background-color: #ccc;
    .about-image{
      background-color: #f00;
    }
    .about-bio{
      background: #efefef;
    }
}
```

Beleza! Todo o processo está feito e bem explicado. Contudo, o grande "pulo do gato" nessa história está em fazer a sua segunda box se comportar como a primeira. Como isso seria possível? Gostaria de lhe apresentar o recurso **extend** do Sass. Para explicar sua função, imagine que o Bootstrap possua a col **.col-lg-6** e você precise utilizá-la novamente. O **extend** atua como um simplificador de classes nesse sentido. Ficou um pouco confuso? Um exemplo tornará as coisas mais fáceis.

```
.about{
  background-color: #ccc;
  @extend .col-md-9;
  @extend .col-lg-6;
   .about-image{
     background-color: #f00;
     @extend .col-md-6;
   }
   .about-bio{
     background: #efefef;
    @extend .col-md-6;
   }
}
```

Esse aspecto se torna algo muito interessante, pois os códigos não estão duplicados, mas sendo utilizados novamente, o que torna seu projeto ainda mais semântico. Logo, basta aplicar em todos os processos.

Ao salvar e atualizar seu navegador desta vez, você poderá ver que todos os elementos das suas boxes estarão iguais em tamanho, cor e formato. A única diferença está, na realidade, no **.about**, que pode ser excluída. Consegue perceber como todo seu projeto se tornou muito mais consistente e simples, ao ter tudo sendo realizado especificamente para sua aplicação? Detalhe: não há nem uma garantia hoje que você está utilizando o Bootstrap, pois o arquivo utilizado em toda esta operação se chama **style.scss**.

Ou seja, você está oferecendo um projeto personalizado, direcionando seu HTML de forma clara. É só olhar para o CSS utilizado e você verá todas as personalizações inseridas, como as cores, além das extensões do Bootstrap sendo utilizadas internamente. Lembra quando mencionei o tal “pulo do gato”? Você compreende a capacidade de desenvolver um projeto cada vez mais profissional com o Bootstrap, tornando toda a linguagem mais prática. Pronto!

É importante frisar que todo seu ***core*** e desenvolvimento criado sempre está na pasta **source**, que garante controle total sobre sua aplicação, com o projeto que será levado para produção estando na **dist**.

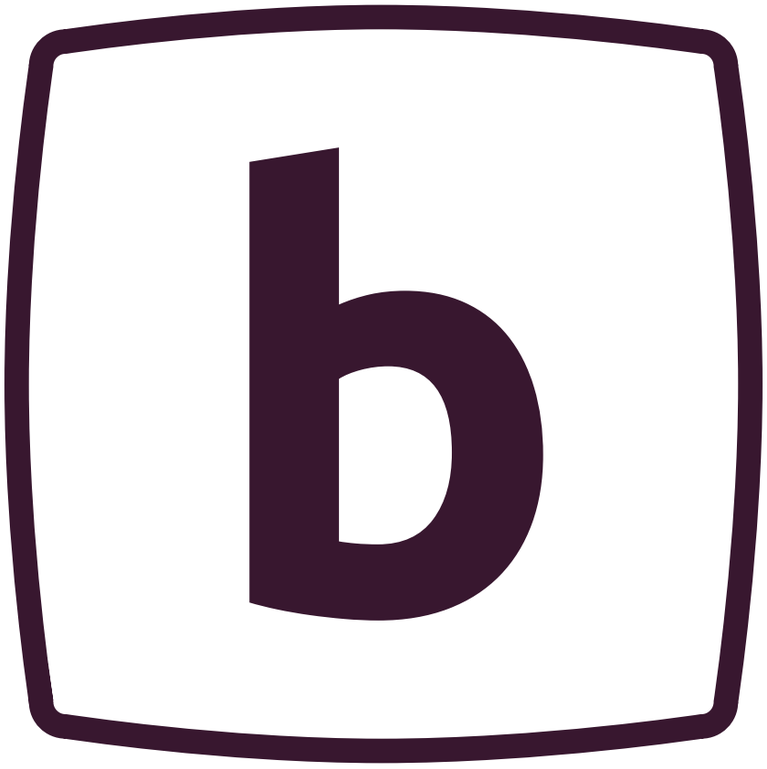
{CONCLUSÃO}

Como você pôde perceber ao longo de toda nossa jornada, a vida de desenvolvedor realmente não é fácil. São muitas demandas e pouco espaço na memória (RAM) para lembrar de inúmeros detalhes. Contudo, a tecnologia que tanto complica nossas vida também pode nos oferecer ferramentas incríveis para agilizar nosso processo de trabalho.

Esse eBook se tratou apenas de uma pequena parcela de todo o material que abordo ao longo de meu curso. Juntos, pudemos entender melhor como pequenos processos, que parecem tão pequenos diante do resto de nossas atividades, podem ser aqueles que mais nos causam atrasos. Mas, ao apresentar extensões, ferramentas e plugins que otimizam essas atividades, o que está realmente em jogo não é apenas o tempo e como o utilizamos.

Na realidade, o que procurei trazer aqui foi a otimização do profissional e de seus potenciais. Ao realizar esses processos, você também está evoluindo enquanto desenvolvedor e aprendendo a tornar seus projetos em algo único, apresentando seus elementos como alguém que domina verdadeiramente os conhecimentos de sua função.

Espero que tenham gostado do material e das dicas! Caso deseje conhecer ainda mais sobre o Bootstrap e suas diversas vantagens, gostaria de lhe convidar para meu curso **Certificação Bootstrap**, onde sei que muitas portas que já foram abertas agora, com nosso eBook, serão destrancadas totalmente e te levarão para um novo caminho em sua carreira.


b