



# Desenvolvimento Front-end em 2020

Uma passagem sobre boas práticas e  
sobre como não odiar CSS



Express

JS





# João Marcus de Lemos Fernandes

## Desenvolvedor Full-stack @ Working Minds

**Github:** @joaomarcuslf2

**Twitter:** @joaomarcuslf

**LinkedIn:** [www.linkedin.com/in/joaomarcuslf/](https://www.linkedin.com/in/joaomarcuslf/)

**Medium:** <https://medium.com/@joaomarcuslf/>

# Tópicos a serem discutidos

## CSS

1. Scss / Less
2. Flexbox
3. Organização

## JS

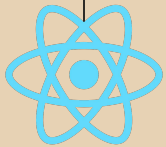
1. Organização
2. Libs úteis
3. Frameworks

## Dicas

1. Dicas de CSS
2. Dicas de JS

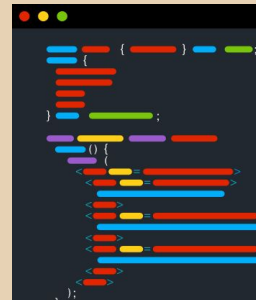
## Etc

Tópicos adicionais  
Referências utilizadas



# CSS

O inferno na terra?



## SCSS / LESS

- São os **pré-processadores** mais conhecidos e utilizados.
- Possuem alguns recursos que facilitam um CSS mais estruturado:
  - **Variáveis;**
  - Funções / Mixins;
  - Herança de classes;
  - **Alinhamento de estilos;**
  - Loops;
- Dependem de um task-runner(**Gulp, Grunt**), ou um pre-run build(**Webpack**).

{.jsx}  
{.scss}  
{.ts}  
{.less}



# Flexbox

- Tem sido adotado como novo padrão de construção de interfaces responsivas.
- Trabalha com o modelo de **layout fluído**, e não no modelo de **blocos**.
- É possível trabalhar com Flex em diversas **direções**.  
(horizontal, vertical, da esquerda para direita, e da direita para esquerda).

## Para referência:

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

## Para aprender:

<https://flexboxfroggy.com/>



# Como organizar CSS

`{.esb}`  
`{.less}`  
`{.vue}`

## OOCSS

### Por quê usar?

Aplica ideias já testadas no desenvolvimento de software como reaproveitamento, o que facilita manutenção.

### Onde usar?

Projetos normalmente de pequeno ou médio porte onde o projeto é estruturado por componente, ou por elemento de tela.

## BEM

### Por quê usar?

BEM é uma metodologia completa de CSS, possui regras bem definidas e um ótimo ferramental.

### Onde usar?

Projetos baseados em templates, projetos médios ou grandes, que possuam uma equipe consideravelmente grande ou multidisciplinar.

## ITCSS

### Por quê usar?

ITCSS é uma metodologia bem estruturada e bem organizada, e é bem versátil para aplicar outras metodologias dentro dele.

### Onde usar?

Projetos no estilo Tema, que possuam variáveis, e que podem ser bem reaproveitáveis.

# OOCSS - Overview

- **OOCSS** se propõe a **resolver** alguns problemas comuns de CSS:
  - Dificuldade de manutenção.
  - Tamanho dos arquivos CSS.
  - Repetição de propriedades comuns.
  - Fragilidade dos estilos por conta da herança em cascata.

## Desvantagens:

- O estilo se torna dependente dos elementos em tela.
- O OOCSS resolve muito bem os componente, porém peca um pouco na hora de fazer a estrutura mais genérica da página(tema, templates).

`{.jsx}`  
`{.scss}`  
`{.ts}`  
`{.less}`





# OOCSS - Exemplos

```
.navbar-element {  
  @extend .vertical-center-container;  
  @extend .horizontal-center-container;  
  
  height: 10vh;  
  padding: 10px;  
  background-color: rgba(100, 200, 20, .1);  
}
```

```
.horizontal-center-container {  
  @extend .flex-container;  
  justify-content: center;  
}
```

```
.vertical-center-container {  
  @extend .flex-container;  
  align-items: center;  
}
```

```
✓ sass  
  ✓ bases  
    flex-container.scss  
    horizontal-center-conta...  
    vertical-center-contain...  
  ✓ objects  
    content-element.scss  
    footer-element.scss  
    navbar-element.scss
```

```
.flex-container {  
  display: flex;  
  width: 100%;  
}
```

# BEM - Overview

`{.jsx}`  
`{.scss}`  
`{.ts}`  
`{.less}`

Alguns dos problemas que o **BEM** resolve:

- Desacoplamento do código.
- Falta de padrão no CSS.

○ **BEM** utiliza um padrão de nomenclatura:

- **B**loco: Elemento mais abstrato da tela, normalmente o container.
- **E**lemento: O elemento que deseja-se alcançar.
- **M**odificador: Um estado do elemento ou do bloco.

## Desvantagens:

- O BEM torna os seletores bem grandes, e pouco intuitivo para pessoas que não estão acostumadas.
- Precisa-se de uma estrutura para ser bem aplicado.



# BEM - Exemplos

```
<nav class="navbar">
  <div class="navbar__brand">
    Logo
  </div>

  <div class="navbar__links">
    <a class="navbar__link--active" href="#">
      Nav 1
    </a>

    <a class="navbar__link" href="#">
      Nav 2
    </a>

    <a class="navbar__link" href="#">
      Nav 3
    </a>

    <a class="navbar__link" href="#">
      Nav 4
    </a>
  </div>
</nav>
```

```
.navbar {
  display: flex;
  align-items: center;
  justify-content: space-between;

  &__brand {
    display: flex;
    justify-content: center;

    &__links {
      display: flex;

      &__link {

        &--active {
          text-decoration: underline;
        }
      }
    }
  }
}
```

# ITCSS - Overview

`{.jsx}`  
`{.scss}`  
`{.ts}`  
`{.less}`

## Vantagens de usar:

- Organiza o código do CSS aplicando o grosso do código nas camadas mais genéricas.
- Facilmente reutilizável entre projetos, mudando apenas as variáveis.
- O ITCSS é bem versátil, podendo ser aplicado parcialmente, ou aplicado em conjunto com OOCSS, e BEM.

## Desvantagens:

- Precisa-se de uma estrutura para ser bem aplicado.
- Bem complexo para projeto pequenos.



# ITCSS - Estrutura



- **Ferramentas:** funções e mixins
- **Genérico:** estilos genéricos (resets, normalizes etc.)
- **Elementos:** estilização de elementos HTML diretamente
- **Objetos:** padrões não cosméticos
- **Componentes:** peças de UI isoladas (menu dropdown, carousel etc.)
- **Trumps:** utilitários, helpers, sobrescrita de classes;





# ITCSS - Exemplos

```
/*  
  Using ITCSS Structure in imports  
*/
```

```
@import  
  'settings/settings',  
  'tools/tools',  
  'generic/generic',  
  'elements/elements',  
  'objects/objects',  
  'components/components',  
  'trumps/trumps';
```

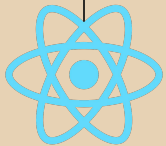
- ▼ sass
  - > components
  - > elements
  - > generic
  - > objects
  - > settings
  - > tools
  - > trumps

```
ls node_modules/bulma/sass -la
```

```
base  
components  
.DS_Store  
elements  
form  
grid  
helpers  
layout  
utilities
```

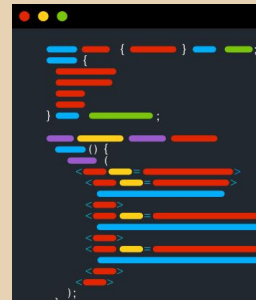
```
// Import only what you need from Bulma
```

```
@import "../node_modules/bulma/sass/utilities/_all.sass";  
@import "../node_modules/bulma/sass/base/_all.sass";  
@import "../node_modules/bulma/sass/elements/button.sass";  
@import "../node_modules/bulma/sass/elements/container.sass";  
@import "../node_modules/bulma/sass/elements/title.sass";  
@import "../node_modules/bulma/sass/form/_all.sass";  
@import "../node_modules/bulma/sass/components/navbar.sass";  
@import "../node_modules/bulma/sass/layout/hero.sass";  
@import "../node_modules/bulma/sass/layout/section.sass";
```



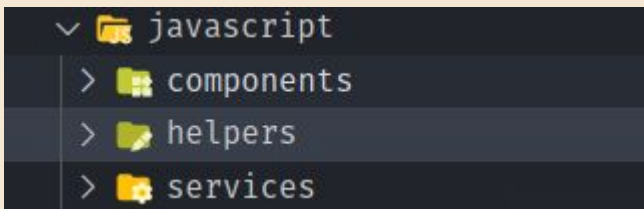
# JS

Uncaught TypeError: Cannot read property 'message' of undefined

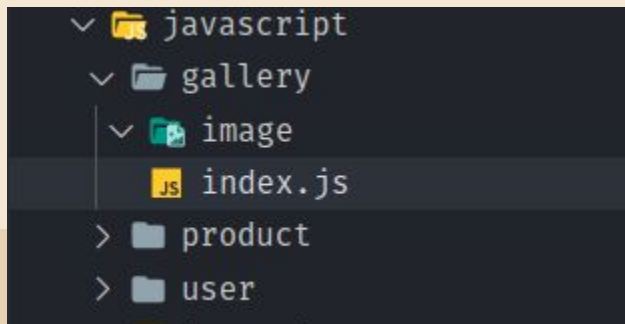


## Organização - Agnósticos:

- **Domain Driven Design(DDD):** Trabalha organizando a estrutura de arquivos pela sua **responsabilidade**, aplicado muito em APIs, exemplo:



- **Model Driven Architecture(MDA):** Divide a estrutura de arquivos usando um namespace referente ao **modelo** que aquele arquivo pertence, exemplo:






## Organização - Não-Agnósticos:

- **Single Page Component:** Padrão aplicado ao VueJS, divide os componentes em um arquivo único contendo em blocos separados o template, o css e o js. Normalmente é considerado produtivo para projetos pequenos, ou prototipação.
- **Container + Component Split:** Padrão aplicado ao Redux, divide a lógica de receber e gerenciar os estados em arquivos de contêiner, e deixa os componentes com a responsabilidade de consumir esse estado e exibir a camada de visualização,



## Libs Úteis

- **Moment.js:** Lib responsável por resolver datas e formatos de datas tanto no Node.js, quanto no Cliente.
  - **Lodash:** Antigamente chamado de underscore.js, aplica diversas funções úteis com boa performance e são extremamente úteis.
  - **FetchAPI:** Esse já não é uma lib e sim uma especificação de Requests HTTP já padronizada com o uso de Promises.
  - **Redux:** Mais utilizado em conjunto com o React, aplica conceitos de estados imutáveis tornando o controle de estado muito mais prático.
  - **RxJS:** Lib que proporciona diversas funções de controle de eventos usando o modelo de Observable.
- 



# Frameworks

- **React:** Também chamado de Lib, aplica uma abordagem modular onde o React se propõe a resolver a camada de View, e permitindo um uso de JS mais próximo do puro, utiliza uma linguagem JSX para construção das telas.
- **Angular:** Framework destinado a prover diversas ferramentas para que o desenvolvedor se preocupe com a implementação das regras de negócio. De todos os citados é o mais opinativo e é indicado o uso de Typescript para melhor segurança do código.
- **VueJs:** Aplica um pouco dos dois exemplos citados acima, o VueJS oferece um ferramental, organização e linguagem de template parecidas com o Angular, porém aplica a ideia do React de resolver bem View e o dev é o responsável pela maior parte do código. É o que menos expressividade entre os três citados.



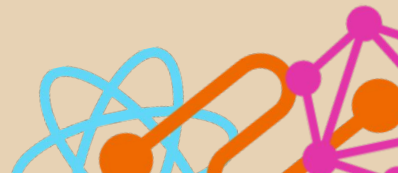


# Dicas

Alguns atalhos para uma vida mais feliz

CSS

JS





## Dicas de CSS - Úteis

- **Fórmula para os Seletores:**

Quando tiver com problemas de especificidade de um seletor aplica a regra:

**ID Classe Elemento Genérico**

Vê o exemplo dos dois seletores

<code>.nav-item.is-active &gt; img</code>	<b>0 2 1 0</b>
<code>.nav-item.is-active &gt; *</code>	<b>0 2 0 1</b>
<code>#nav img</code>	<b>1 0 1 0</b>

O de baixo é mais específico pois pontua 1010 e o de cima 210, e o do meio é o menos específico com apenas 201.

- **Pseudo classes de estado :not(.class) e :nth-child(n):**

Com ambos você consegue aplicar estilos em elementos utilizando queries.

Utilizando o exemplo anterior;

```
.nav-item:not(.is-active)
#nav .nav-item:nth-child(1)
```





## Dicas de CSS – Gambiarras

- **Positions absolute dentro de relative:**

Colocando os elemento com “**position: absolute**” dentro de um elemento com “**position: relative**” você tem controle de poder posicionar o elemento dentro de um espaço controlado.

```
.wrapper {  
  position: relative;  
  height: 500px;  
}
```

```
.wrapper .element {  
  position: absolute;  
  top: 0;  
}
```

- **\_ na frente de propriedades para o IE:**

O IE é o único browser que lê propriedades CSS com \_, então caso você queira criar estilos específicos para o IE algumas propriedades suportam \_ na frente.

```
.element {  
  width: 500px;  
  _width: 500px  
}
```





## Dicas de JS - Úteis

- **Funções para transformar dados:**
  - **Map:** Recebe um array e retorna um array, porém transforma individualmente cada elemento do array.
  - **Filter:** Recebe um array e retorna um array, porém permite apenas os itens que correspondam à condição que você especificar.
  - **Reduce:** Recebe um array e retorna a estrutura que desejar, recebe uma callback com o dado transformado pela última interação e o elemento anterior.
- **Convertendo em boolean puro com !!:**
  - Você pode transformar valores falsey e valores truth em booleanos puros como true e false colocando !! na frente da variável.

```
1  var a = 0;  
2  
3  var boolA = !!a;
```





## Dicas de JS - Úteis

- **Default values:** Funções permitem que você as invoque com valores padrões caso o dev deseje.

```
function callWS(url, options = {}, method = "GET") {  
  /* ... */  
}
```

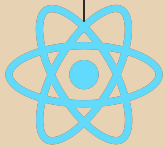
- **Promise.all recebendo valor fixo:** É possível passar em um promise.all valores fixos.

```
User  
  .getUserData()  
  .then((data) => {  
    return Promise.all([  
      data,  
      Products.getCartFromUser(user.id),  
      Message.getMessagesFromUser(user.id),  
    ])  
  })
```

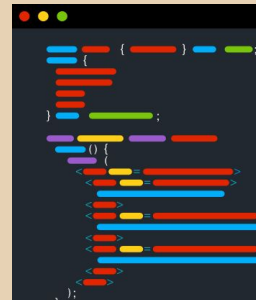
```
    .then([  
      userData,  
      productsData,  
      messagesData  
    ]) => {  
      /* ... */  
    })
```







**Etc**



## Tópicos Adicionais



- **Webpack:** Atualmente a principal ferramenta para transpilar, buildar e empacotar os assets.
- **Typescript:** Superset de JS, aplica recursos de OO e de tipagem estática à linguagem.
- **Eslint:** Principal ferramenta para formatar código e criar um padrão de código.
- **VS Code:** Principal editor para desenvolvimento JS, já vem por padrão equipado com diversos recursos para facilitar o desenvolvimento.
- **HTML Semântico:** Uma forma de escrita de HTML, enriquece o HTML com propriedade informativas, essencial para Acessibilidade.
- **Node.js:** Um módulo que permite o desenvolvimento de CLIs, APIs, Desktop apps utilizando apenas JS.



# Referências utilizadas

Tárcio Zemel, livro **“CSS Eficiente”**, editora Casa do Código.

**Bulma.io**, “<https://bulma.io/documentation/customize/with-node-sass/>”

**A Complete Guide to Flexbox**, “<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>”

**SASS Lang**, “<https://sass-lang.com/>”

**Less.js**, “<http://lesscss.org/usage/>”

**Moment.js**, “<https://momentjs.com/>”

**RxJS**, “<https://rxjs-dev.firebaseapp.com/>”

**Fetch API**, “[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)”

**Lodash**, “<https://lodash.com/>”





## Referências utilizadas

**CSS Pseudo-classes**, "[https://www.w3schools.com/css/css\\_pseudo\\_classes.asp](https://www.w3schools.com/css/css_pseudo_classes.asp)"

**Domain Driven Design vs Model Driven Architecture**,  
"<https://stackoverflow.com/questions/4166816/domain-driven-design-vs-model-driven-architecture>"

**Comparison of Domain-Driven Design and Clean Architecture Concepts**,  
"<https://khalilstemmler.com/articles/software-design-architecture/domain-driven-design-vs-clean-architecture/>"

**12 Extremely Useful Hacks for JavaScript**, "<https://blog.iscrambler.com/12-extremely-useful-hacks-for-javascript/>"

**Css Hacks – Ruim com eles, pior sem eles**, "<https://tableless.com.br/csshacks/>"





**Fim! :D**

