

Arquitetura de css

Samuel Martins

Arquitetura de css

- CSS importa, e muito!
- Um CSS mal feito prejudica toda a manutenção de um projeto web;
- Refatorar CSS não é **NADA** fácil;
- Já precisou de fazer alguma refatoração de CSS em projeto web?

Refatoração em arquivo TypeScript

```
3 references
25 export class TypeScriptVersionPicker {
26     6 references
27     private _currentVersion: TypeScriptVersion;
28
29     1 reference
30     public constructor(-
31     ) {-
32     }
33
34     1 reference
35     public async show(firstRun?: boolean): Promise<{ oldVersion?: TypeScriptVersion, newVersion?:
36         const pickOptions: MyQuickPickItem[] = [];
37
38         pickOptions.push({
39             label: localize('learnMore', 'Learn More'),
40             description: '',
41             id: MessageAction.learnMore
42         });
43
44         const shippedVersion = this.versionProvider.defaultVersion;
45
46         pickOptions.push({
47             label: (!this.useWorkspaceTsdkSetting
48
49
50
51
52
53
54
```

Arquitetura de CSS

- CSS está ligado diretamente a performance:
 - Override de estilos é custoso para o browser;
 - Evitar o uso de *!important* diminui o número de re-render na fase de “painting”;
 - Existem consultorias de performance especializadas em web: <https://3perf.com/content>;
- CSS está diretamente ligado à escala:
 - Manter a consistência de estilos entre as telas é de suma importância;
 - Editar estilos antigos não deve ser uma tarefa difícil;
 - Reutilizar estilos já criados é extremamente importante.

Arquitetura de CSS

- Colisão e nome de classes:
 - Ao dividir em múltiplos arquivos, vamos decorar todos os nomes para evitar de repeti-los em arquivos diferentes?
 - O nome das classes e ID dizem respeito ao que realmente fazem?

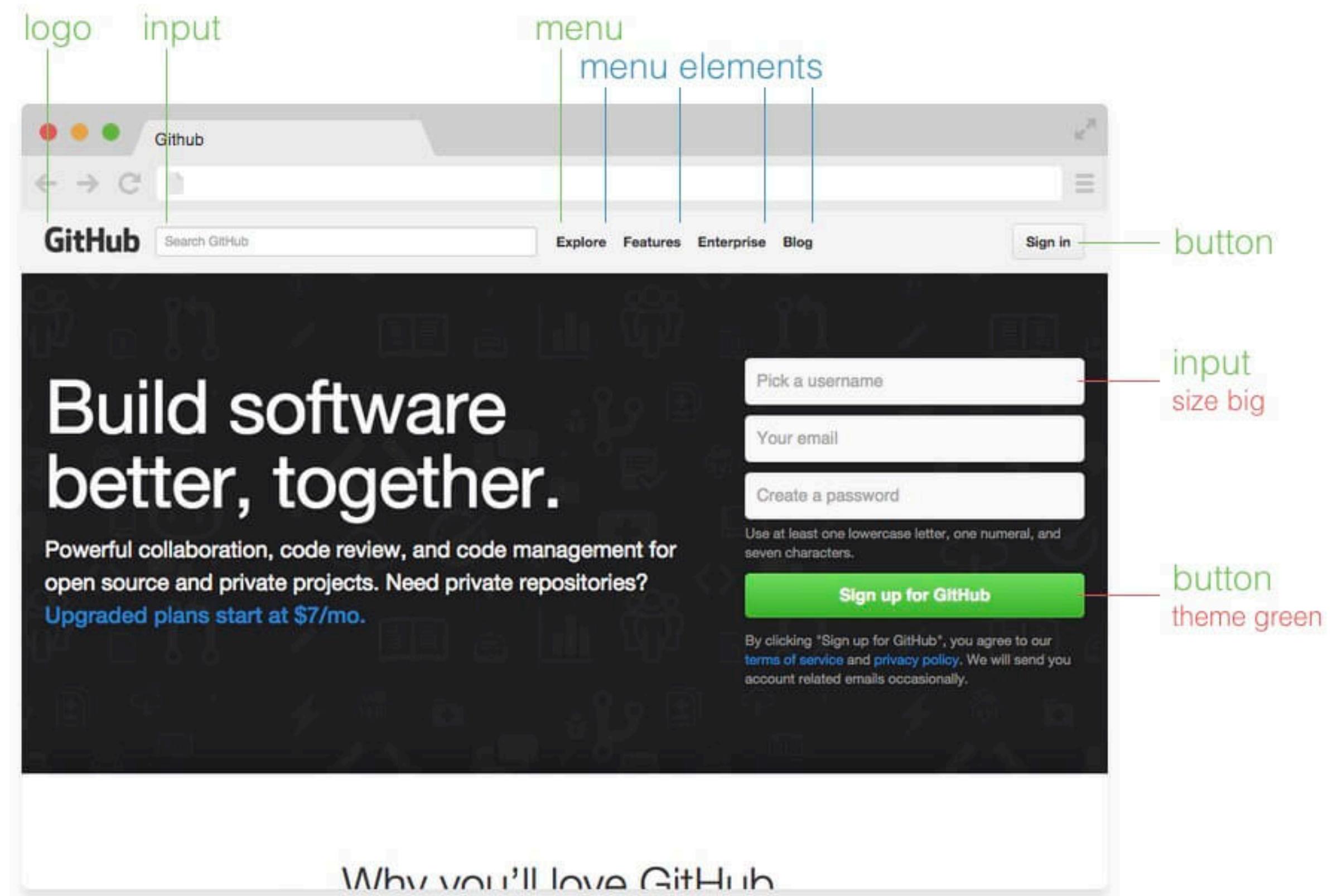
Arquitetura de CSS

- Algumas style-guides e metodologias tentam resolver tais problemas:
 - BEM: *Block - Element - Modifier*;
 - OOCSS: *Object-Oriented CSS*;
 - CSS Funcional: Tachyons e Tailwind;
 - CSS-in-JS;

Block - Element - Modifier (BEM)

- Block:
 - Exemplos: header, container, menu, checkbox, input;
- Element:
 - Exemplos: ítem de menu, ítem de lista, título de cabeçalho;
- Modifier: estado em que o elemento ou bloco se encontra
 - Exemplos: disabled, active, fixed, background cinza;

Block - Element - Modifier (BEM)



Block - Element - Modifier (BEM)

```
1 <nav class="main-menu">
2   <ul class="main-nav">
3     <li class="main-nav__item main-nav__item--is-
active">Home</li>
4     <li class="main-nav__item">Sobre</li>
5     <li class="main-nav__item">Contato</li>
6   </ul>
7 </nav>
```

Block - Element - Modifier (BEM)

```
1 <nav class="main-menu">
2   <ul class="main-nav" style="list-style-type: none;">
3     <li class="main-nav__item main-nav__item--is-
active">Home</li>
4     <li class="main-nav__item">Sobre</li>
5     <li class="main-nav__item">Contato</li>
6   </ul>
7 </nav>
```

Block - Element - Modifier (BEM)

```
1 <nav class="main-menu">  
2   <ul class="main-nav">  
3     <li class="main-nav__item main-nav__item--is-  
active">Home</li>  
4     <li class="main-nav__item">Sobre</li>  
5     <li class="main-nav__item">Contato</li>  
6   </ul>  
7 </nav>
```

Element



Block - Element - Modifier (BEM)

```
1 <nav class="main-menu">  
2   <ul class="main-nav">  
3     <li class="main-nav__item main-nav__item--is-active">Home</li>  
4     <li class="main-nav__item">Sobre</li>  
5     <li class="main-nav__item">Contato</li>  
6   </ul>  
7 </nav>
```

Modifier



Block - Element - Modifier (BEM)

```
1 /* Block */
2 .main-nav {
3   margin-bottom: 10px;
4 }
5
6 /* Element */
7 .main-nav__item {
8   padding: 20px;
9   color: black;
10 }
11
12 /* Modifier */
13 .main-nav__item--is-active {
14   border: 1px solid black;
15 }
```

Block - Element - Modifier (BEM)

Home

Sobre

Contato

Block - Element - Modifier (BEM)

- Convenção já definida para nomenclatura de classes;
- Visão clara de estilos para elementos filhos;
- Visão clara de estados;
- Nome de classes muito grandes;
- <http://getbem.com/naming/>;

Object-oriented CSS (OOCSS)

- Separação entre CSS de estrutura e *skin*;
- Structure properties:
 - *Width | Height | Padding | Margins | Overflow;*
- Skin:
 - *Color | Background | Font | Shadow.*

Object-oriented CSS (OOCSS)

```
1 .button {  
2     width: 150px;  
3     height: 50px;  
4     background: #FFF;  
5     border-radius: 5px;  
6 }  
7  
8 .button-2 {  
9     width: 150px;  
10    height: 50px;  
11    background: #000;  
12    border-radius: 5px;  
13 }
```

Object-oriented CSS (OOCSS)

```
● ● ●  
  
.skin-1 {  
    background: white;  
}  
  
.skin-2 {  
    background: black;  
}  
  
.button {  
    width: 50px; height: 50px;  
    border-radius: 5px;  
}
```

Object-oriented CSS (OOCSS)



```
<button class="button skin-1">Botão</button>
```

Object-oriented CSS (OOCSS)

- Separação entre container e conteúdo;
- Ausência de classes “herdadas”.

Object-oriented CSS (OOCSS) - Antes

```
1 #sidebar {  
2     padding: 2px;  
3     left: 0;  
4     margin: 3px;  
5     position: absolute;  
6     width: 140px;  
7 }  
8  
9 #sidebar .list {  
10    margin: 3px;  
11 }  
12  
13 #sidebar .list .list-header {  
14     font-size: 16px;  
15     color: red;  
16 }
```

Object-oriented CSS (OOCSS) - Depois

```
1 .sidebar-container {  
2     position: absolute;  
3     padding: 2px;  
4     margin: 3px;  
5     left: 0;  
6 }  
7  
8 .sidebar {  
9     width: 140px;  
10 }  
11  
12 .list {  
13     margin: 3px;  
14 }  
15  
16 .list-header {  
17     font-size: 16px;  
18     color: red  
19 }
```

CSS funcional

- Ideia de ter micro classes de CSS para compor um elemento;
- Cada classe altera uma única propriedade do CSS;
- Classes sempre tratam responsividade desde a concepção;
- Exemplos:
 - [Tachyons](#) | [Tailwind CSS](#) | [BassCSS](#)

CSS funcional - tachyons - propriedades de texto

```
1 .tl { text-align: left; }  
2 .tr { text-align: right; }  
3 .tc { text-align: center; }  
4 .tj { text-align: justify; }
```

CSS funcional - tachyons - propriedades de largura

```
1 .w-10 { width: 10%; }  
2 .w-20 { width: 20%; }  
3 .w-25 { width: 25%; }  
4 .w-30 { width: 30%; }
```

CSS funcional - tachyons - espaçamentos

```
1 :root {  
2   --spacing-none: 0;  
3   --spacing-extra-small: .25rem;  
4   --spacing-small: .5rem;           ← CSS variables  
5   --spacing-medium: 1rem;  
6 }  
7  
8 .pa0 { padding: var(--spacing-none); }  
9 .pa1 { padding: var(--spacing-extra-small); }  
10 .pa2 { padding: var(--spacing-small); }  
11 .pa3 { padding: var(--spacing-medium); }
```

CSS funcional - tachyons - utilização no html

```
1 <div class="w-10 tl pa1">  
2   Conteúdo do elemento aqui  
3 </div>
```

CSS funcional

- **Responsividade:** todas as classes possuem tratamento responsivo;
- **Reutilizável e modular:** inúmeras classes prontas para serem usadas em qualquer tela;
- **Legibilidade:** ao olhar o HTML, você sabe exatamente qual será o seu estilo;
- **Produtividade:** escrever HTML é mais rápido, pois diminui a necessidade de criar classes para modificar propriedades simples;

CSS-IN-JS

- Formato de escrita de **CSS em arquivos JavaScript**;
- Classes CSS podem ser escritos em **formato de função**, que são traduzidos em componentes posteriormente;
- **CSS dinâmico**: uma classe CSS pode se comportar de forma diferente de acordo com os parâmetros que são passados.

CSS-IN-JS - Header

```
1 import styled from 'styled-components'  
2  
3 export const Header = styled.header`  
4   background: black;  
5   color: red;  
6   font-size: 14px;  
7 `;
```

CSS-IN-JS - Header

```
// JSX
<div>
  <Header> /* Atenção ao H maiúsculo */
  Conteúdo do cabeçalho aqui
  </Header>
</div>
```

CSS-IN-JS - Button

```
1 import styled from 'styled-components'  
2  
3 const Button = styled.button`  
4   background: ${props => props.primary ? "green" :  
"white"};  
5   color: ${props => props.primary ? "white" : "green"};  
6 `;
```

CSS-IN-JS

```
// JSX
<div>
  <Button primary>Conteúdo do botão</Button>
</div>
```

CSS-IN-JS

- Suporte a poucos frameworks. React concentra as maiores opções:
 - React: styled-components e emotion;
 - Vue: Vue-styled-components;
 - Angular: possui mecanismo próprio - Component Styles.
- Numa árvore de elementos DOM/componentes, fica difícil visualizar o que é componente e o que é estilo

Pré-processadores de css

Samuel Martins

Pré-processadores de css

- Como o próprio nome diz: um **pré-processador**, com a capacidade de adicionar novos recursos na escrita de estilos;
- Três principais ferramentas com o mesmo propósito: **SASS**, **LESS** e **Stylus**;
- Permite gerar CSS baseado na linguagem do pré-processador.

Pré-processadores de css

- Três principais recursos:
 - Variáveis;
 - Mixins;
 - Escrita de estilos por herança;

Pré-processadores de css - HTML

```
1 <ul class="main-nav">
2   <li class="nav-item">Link 1</li>
3   <li class="nav-item">Link 2</li>
4   <li class="nav-item">Link 3</li>
5 </ul>
```

Pré-processadores de css - CSS Puro

```
1 .main-nav {  
2   background: green;  
3 }  
4  
5 .main-nav .nav-item {  
6   color: #fff;  
7 }
```

Pré-processadores de css (SASS)

```
1 .main-nav {  
2   background: green;  
3  
4   .nav-item {  
5     color: #fff;  
6   }  
7 }
```

Pré-processadores de css - variáveis (SASS)

```
1 $primary-bg: 'green';
2 $primary-color: 'white';
3
4 .main-nav {
5   background: $primary-bg;
6
7   .nav-item {
8     color: $primary-color;
9   }
10 }
```

Pré-processadores de css - variáveis - arquivo separado

```
1 // variables.scss  
2 $primary-bg: 'green';  
3 $primary-color: 'white';
```

Pré-processadores de css - variáveis

```
1 @import "variables.scss";  
2  
3 .main-nav {  
4     background: $primary-bg;  
5  
6     .nav-item {  
7         color: $primary-color;  
8     }  
9 }
```

Pré-processadores de css - mixins

```
1 @mixin transform($property) {  
2   -webkit-transform: $property;  
3   -ms-transform: $property;  
4   transform: $property;  
5 }  
6  
7 .box { @include transform(rotate(30deg)); }
```

Pré-processadores de css - mixins

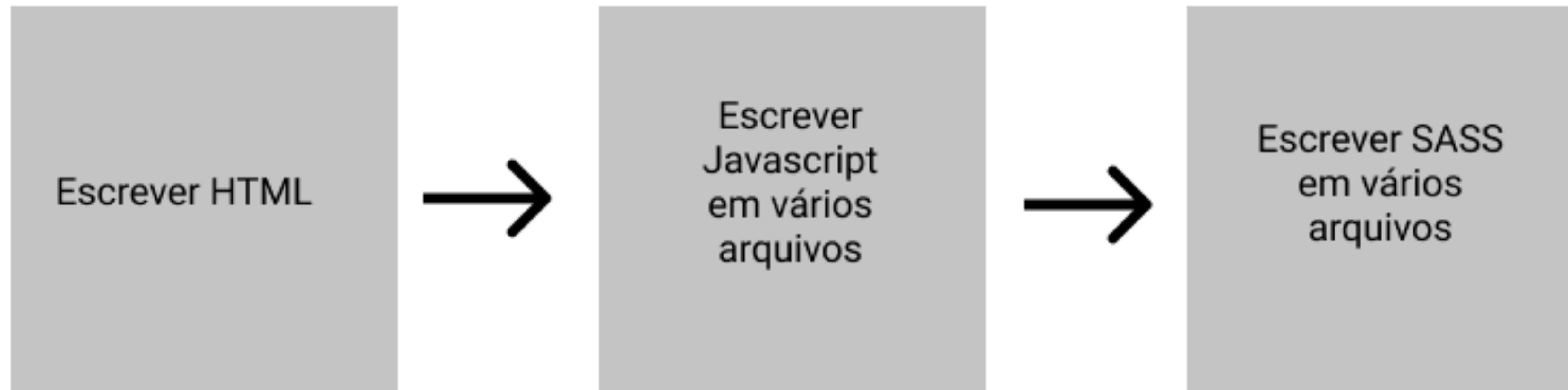
- Permite ter uma boa escalabilidade na criação de recursos compartilhados (cores, fontes, backgrounds...);
- Flexibilidade na criação de mixins: possibilidade de reutilizar muito código;
- Complexidade: necessita de alguma ferramenta de build para interpretar os pré-processadores.

Ferramentas de bundlers e otimização de código

Samuel Martins

Bundlers

- Pipeline de desenvolvimento de código pode ser custoso:
- Em um cenário com uso de uma stack simples podem ser necessárias várias ferramentas para Build do código. Imagine a stack Html, Javascript e Sass:



Bundlers

Escrever HTML

- Escrever o HTML semântico;
- Incluir manualmente todo novo arquivo JavaScript criado;
- Incluir manualmente todo novo arquivo CSS criado;
- Mudar da abordagem de “arquivos” para a abordagem em “bloco” é totalmente inviável;

Bundlers

Escrever
Javascript
em vários
arquivos

- Preocupação com compatibilidade entre browsers. Nem todos os recursos modernos estão disponível em versões mais antigas;
- Incluir manualmente no HTML todo novo arquivo JS criado ou;
- Concatenar e minificar o conteúdo de todos os arquivos manualmente;

Bundlers

Escrever SASS
em vários
arquivos

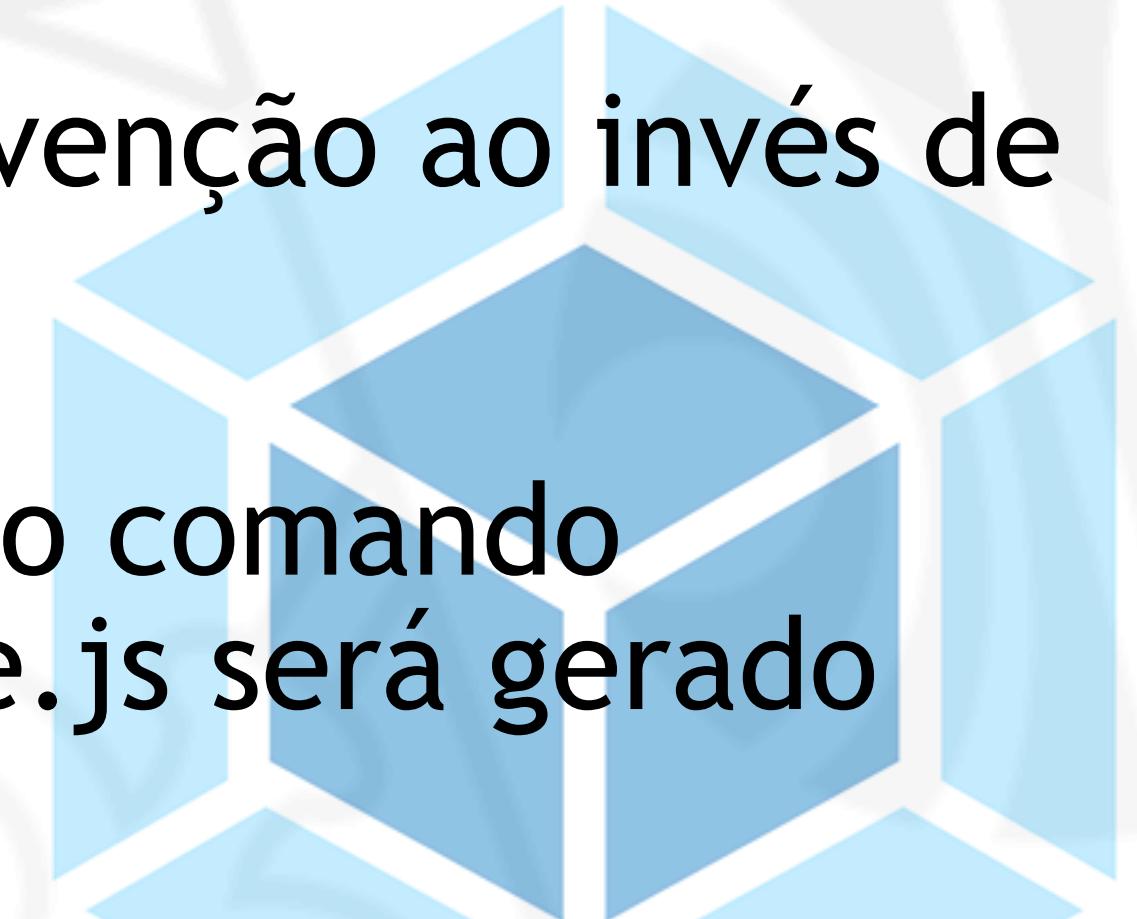
- Compilar o SASS em arquivos CSS;
- Incluir manualmente no HTML todo novo arquivo CSS criado ou;
- Concatenar e minificar o conteúdo de todos os arquivos manualmente;

Bundlers

- Os bundlers foram criados para resolver justamente esse tipo de problema;
- Um arquivo de configuração é necessário para automatizar todas essas tarefas;
- Possibilidade de customizar, a qualquer momento, as tarefas já automatizadas;
- Frameworks mais novos vêm com algum bundler embutido, mas conhecendo o seu funcionamento podemos customizar e adaptar à nossa arquitetura.

Bundlers - Webpack

- Webpack é o bundler mais utilizado pela comunidade;
- Utilizado por Angular, Vue e React;
- Gera automaticamente um grafo de dependências entre os arquivos escritos;
- Utiliza o conceito “*convention over configuration*” - “convenção ao invés de configuração”;
- Por convenção, seguindo a estrutura a seguir, basta rodar o comando “webpack” na linha de comando e um arquivo dist/bundle.js será gerado



Bundlers - Webpack

src/index.js

```
import bar from './bar';  
  
bar();
```

src/bar.js

```
export default function bar() {  
    //  
}
```



Bundlers - Webpack

page.html

```
<!doctype html>
<html>
  <head>
    ...
  </head>
  <body>
    ...
    <script src="dist/bundle.js"></script>
  </body>
</html>
```

Bundlers - Webpack

- Utiliza o conceito de “loaders”, onde é possível adicionar plugins ao Pipeline de build do webpack;
- sass-loader: loader responsável por compilar arquivos sass em css;

Bundlers - Webpack

webpack.config.js

```
module.exports = {
  module: {
    rules: [
      {
        test: /\.s[ac]ss$/i,
        use: [
          // Creates `style` nodes from JS strings
          'style-loader',
          // Translates CSS into CommonJS
          'css-loader',
          // Compiles Sass to CSS
          'sass-loader',
        ],
      },
    ],
  },
};
```

Arquitetura modular para aplicações escaláveis

Samuel Martins

Arquitetura modular para aplicações escaláveis

- Projetos estão sempre em evolução e crescimento;
- Encontrar uma arquitetura ideal logo de cara nem sempre é possível;
- Arquitetura evolutiva: projeto deve ser escalável e flexível o suficiente para poder sofrer alterações no futuro (*last responsible moment*);
- Objetivo: encontrar uma arquitetura que seja agnóstica a frameworks e que possa ser aplicada em qualquer stack.

Arquitetura modular para aplicações escaláveis

- Arquiteturas podem (e devem) ser adaptadas para pequenos e grandes projetos;
- Divisão de uma aplicação em módulos:
 - UI Components;
 - Page Components;
 - Application routes;
 - API Client;
 - State management.

Ui components

- Componentes que contém a lógica visual e de interação com usuário.

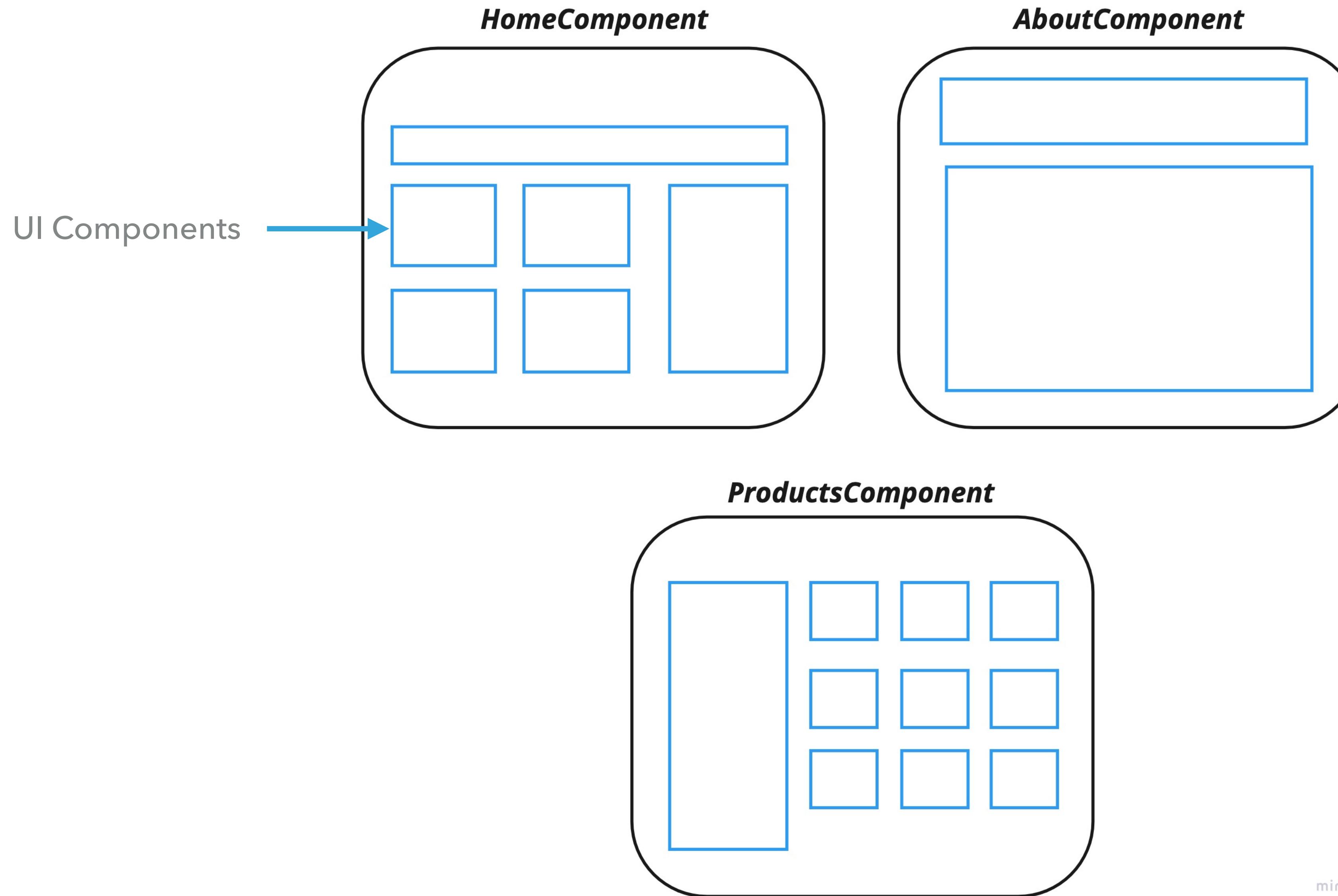
Exemplo:

- ▶ Botões;
- ▶ Menus;
- ▶ Listas;
- ▶ Cabeçalhos;
- ▶ Datepicker.

Page components

- Componentes que representam uma tela e estão associados a uma rota.
Exemplo:
 - ▶ Página principal;
 - ▶ Página de contato;
 - ▶ Listagem de produtos;
 - ▶ Página de configurações;
 - ▶ Página de perfil de usuário.

/ => HomeComponent
/about => AboutComponent
/products => ProductsComponent



Application routes

- Arquivo que contém todas as rotas daquele módulo;
- Toda nova rota (URL) deve estar associado a um componente.

Api Client/Services

- Necessidade de desacoplar a lógica de chamada de APIs da lógica dos componentes;
- Ponto único de contato entre a aplicação e API externa

State management

- Compartilhamento de estados comuns entre os componentes, exemplo:
 - Dados do usuário logado (token, sessão, dados de perfil);
 - Tema visual e sistema de cores (dark mode e etc);
 - Filtro de elementos visuais (ordenação por título, preço e etc).
- SSOT: Single Source of Truth;
- Utilização da arquitetura flux para gerenciamento de estados como exemplo.

Arquitetura modular para aplicações escaláveis - Opção 1

```
~/projects/modular-architecture
$ tree
.
├── api
│   ├── Cart.js
│   ├── Product.js
│   └── User.js
├── components
│   ├── button
│   ├── modal
│   ├── navbar
│   └── tooltip
└── pages
    ├── Home.js
    ├── Login.js
    ├── ProductDetail.js
    └── Products.js
├── routes
│   ├── cart.js
│   ├── index.js
│   └── products.js
└── store
    ├── actions
    └── reducers

11 directories, 10 files
```

Arquitetura modular para aplicações escaláveis - Opção 2

Cart

```
.├── cart
│   ├── api
│   │   └── Cart.js
│   ├── components
│   │   ├── description
│   │   └── shop
│   ├── pages
│   │   └── Cart.js
│   └── Shippment.js
├── routes
│   ├── cart.js
│   └── index.js
└── store
    ├── actions
    └── reducers
```

Products

```
├── products
│   ├── api
│   │   └── Product.js
│   ├── components
│   │   ├── productImageZoom
│   │   └── productItem
│   ├── pages
│   │   └── ProductDetail.js
│   └── Products.js
├── routes
│   ├── index.js
│   └── products.js
└── store
    ├── actions
    └── reducers
```

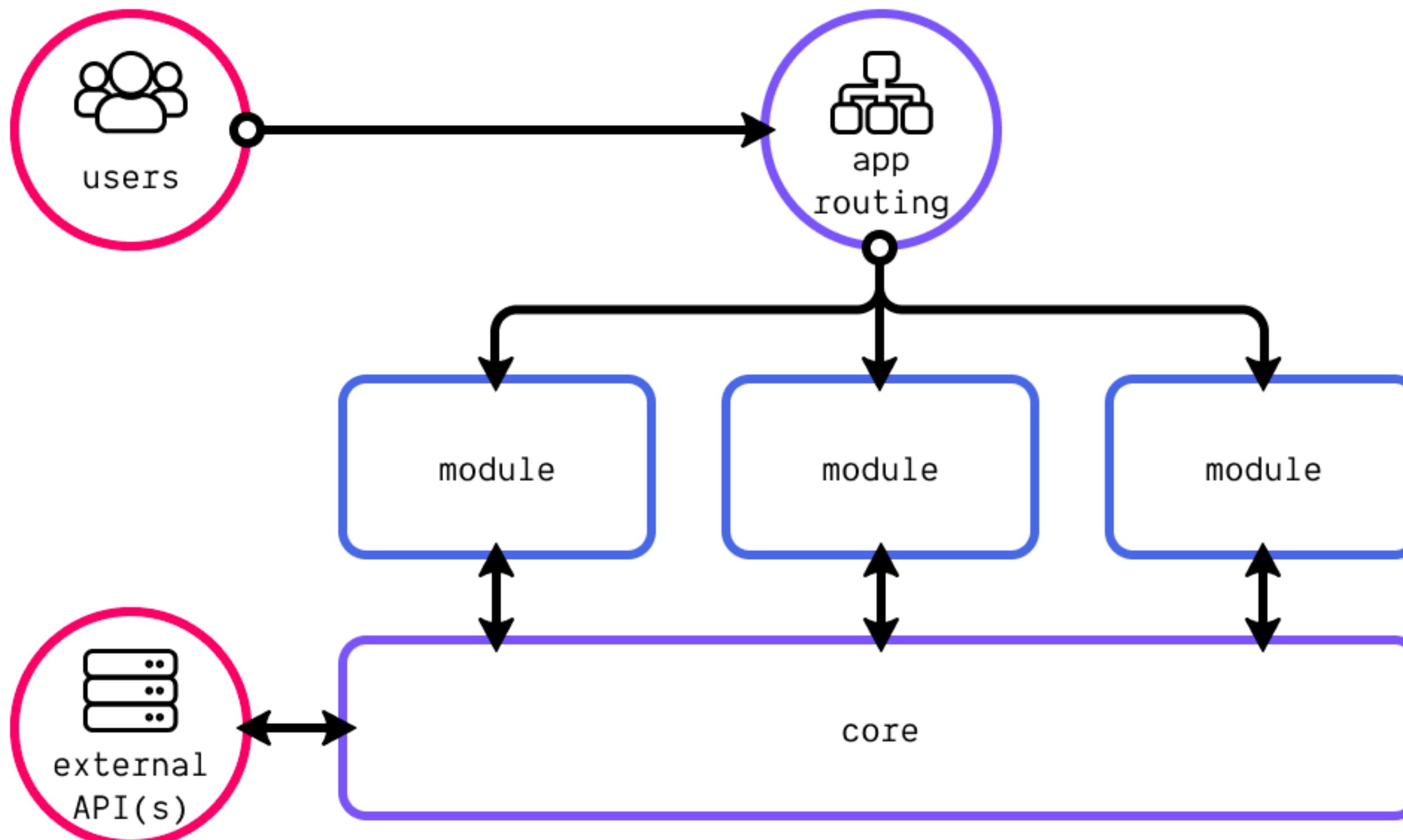
Users

```
└── users
    ├── api
    │   └── User.js
    ├── components
    │   ├── userAvatar
    │   └── userItem
    ├── pages
    │   └── UserDetails.js
    ├── routes
    │   └── index.js
    └── users.js
└── store
    ├── actions
    └── reducers
```

Shared

```
└── shared
    └── components
        ├── button
        ├── modal
        ├── navbar
        └── tooltip
```

Arquitetura - módulo

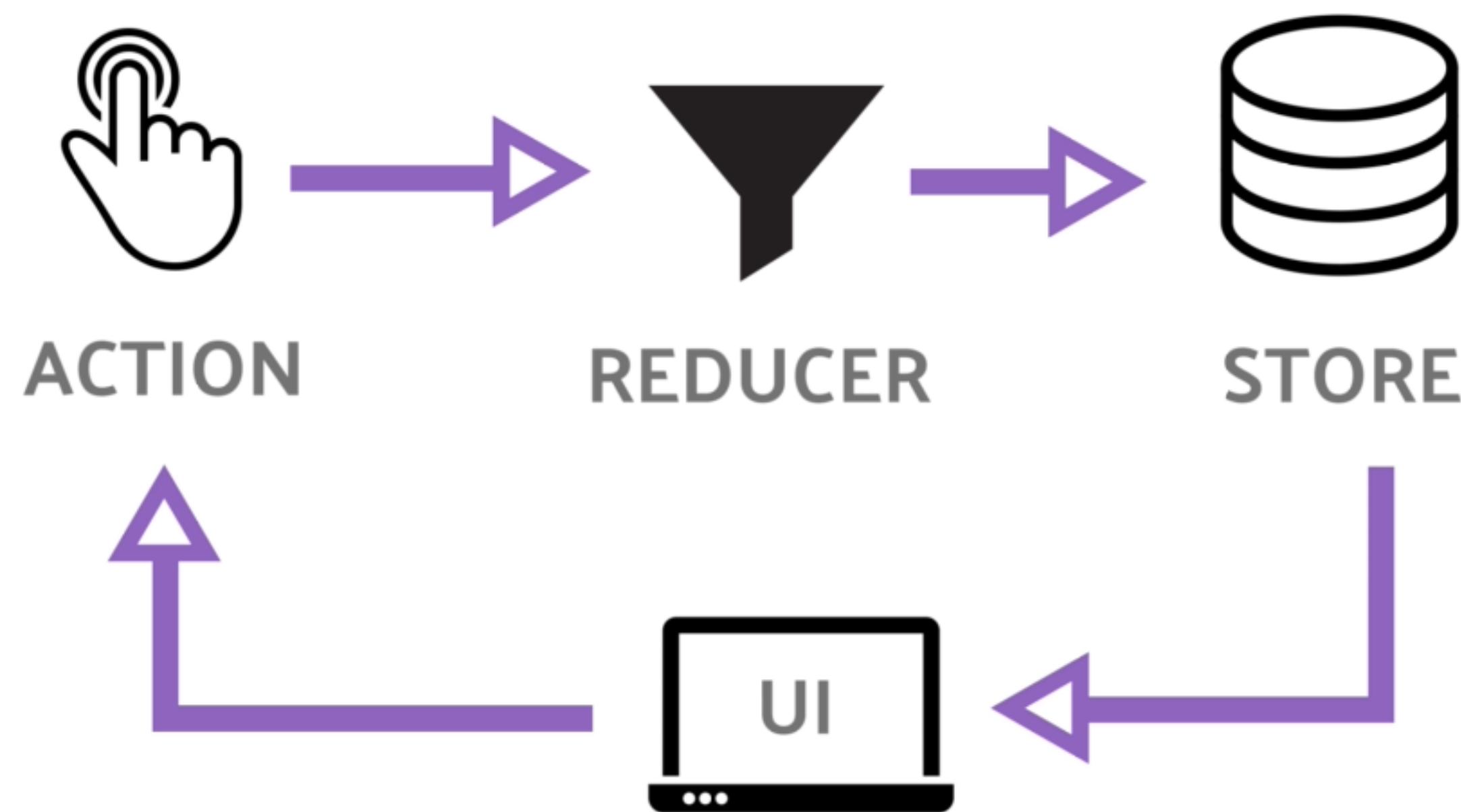


Fonte: <https://kevtiq.co/blog/scalable-front-end-architecture/>

Arquitetura Flux

Samuel Martins

Arquitetura Flux



Fonte: <https://medium.com/reactbrasil/iniciando-com-redux-c14ca7b7dcf>

Arquitetura Flux - store

- Armazena o estado de toda a aplicação;
- Estado é atualizado a partir de algum *dispatcher*;
- Aplicação possui apenas uma única store;
- Comparada a um “estado global”.

Arquitetura Flux - Actions

- Conteúdos enviados da aplicação para a store;
- Descreve o que acontece;
- Precisa ter uma propriedade “*type*”;
- Apenas uma função que retorna um objeto javascript.

Arquitetura Flux - Actions

```
1 const SET_USER = 'SET_USER'  
2  
3 const setUser = user => ({  
4   type: SET_USER,  
5   payload: user,  
6 })
```

Arquitetura Flux - Reducers

- Descreve **como** as alterações acontecem na store e como os dados são transformados;
- Recebe uma action como argumento;
- Funções puras (não alteram nada diretamente fora do seu escopo).

Arquitetura Flux - Reducers

```
● ● ●

1 import { SET_USER } from '../actions';
2
3 function user(state = {}, action) {
4   switch (action.type) {
5     case SET_USER:
6       return {
7         ... state,
8         user: action.payload
9       }
10    default:
11      return state
12  }
13 }
```



PUC Minas
Virtual