

FIAP

FIAP

SLIDER ▢■◀



Sass

O que é

SASS (Syntactically Awesome Stylesheets) é uma extensão de CSS que adiciona funcionalidades poderosas, permitindo escrever estilos de maneira mais eficiente, organizada e reutilizável. Ele é um pré-processador CSS, o que significa que você escreve código SASS que é então compilado para CSS padrão que os navegadores entendem

Documentação: <https://sass-lang.com/documentation/>

Vantagens

Variáveis

Permitem definir valores reutilizáveis como cores, fontes, tamanhos, etc.

Aninhamento

Facilita a escrita de seletores CSS de forma hierárquica, refletindo a estrutura HTML.

Mixins

Blocos de código reutilizáveis que podem ser incluídos em diferentes seletores.

Funções e Operações

Realizar cálculos e manipulações de valores diretamente no CSS.

Partials e Importações

Modularizar o CSS dividindo-o em arquivos menores e mais organizados.

Extensões/Herança

Compartilhar estilos entre diferentes seletores para evitar repetição de código.

SASS vs SCSS

O SASS aceita dois tipos de extensão de arquivo diferentes. No React criamos arquivos, como por exemplo App.css. No SASS isso para para App.sass ou App.scss. Onde:

.sass

Usa uma sintaxe mais simples e sem chaves, com indentação baseada em espaços (lembra Python)

```
1
2  body
3    font: 100% Helvetica, sans-serif
4    color: blue
5    p
6      font-size: 1.5em
7
8  ✨
9
```

.scss

É uma sintaxe mais recente que usa a mesma estrutura do CSS, com chaves {} e ponto-e-vírgula ; tornando-o mais familiar para quem já conhece CSS.

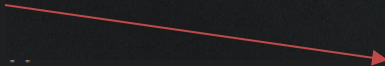
```
1
2  body {
3    font: 100% Helvetica, sans-serif;
4    color: blue;
5    p {
6      font-size: 1.5em;
7    }
8  }
```


Usando SASS

1. Vamos abrir o um projeto já criado com o Vite + React + TypeScript como já vimos em aulas anteriores. Caso não se lembre de como criar, é só executar `npm create vite@latest` e seguir os passos que vimos nas aulas anteriores.
2. Vamos instalar o sass, rodando o comando `npm install sass --save-dev`
3. Então tudo que precisamos fazer é criar um arquivo com a extensão correta. Vamos criar um `App.scss` (ou `.sass` se preferir) e trocar a importação no nosso `.tsx`

App.tsx > App

```
import { useState } from "react"; 4.2k (gzipped: 1.8k)
import reactLogo from "../assets/react.svg";
import viteLogo from "/vite.svg";
import "../App.css";
```



```
1 import { useState } from "react"; 4.2k (gzipped: 1.8k)
2 import reactLogo from "../assets/react.svg";
3 import viteLogo from "/vite.svg";
4 import "../App.scss";
5
```

Usando SASS

Variáveis

Variáveis no SASS/SCSS permitem armazenar valores que você usa repetidamente em seus estilos, como cores, fontes, tamanhos, entre outros. Com variáveis, você pode mudar facilmente esses valores em um único lugar, tornando o código mais fácil de manter e adaptar. Esse recurso foi criado antes da existência das variáveis de CSS. É preferível usar as variáveis de CSS na maioria dos casos, mas é interessante usar as do SASS para alguma manipulação específica

```
$primary: #61dafb;
```

```
p {  
  font-size: 1.5em;  
  color: $primary;  
}
```

Usando SASS

Aninhamento (Nesting)

Aninhamento em SASS/SCSS permite que você escreva estilos de forma hierárquica, refletindo a estrutura do HTML. Isso facilita a leitura e a organização do código, especialmente quando você está lidando com seletores que têm uma relação de hierarquia clara. Usamos o `&` para concatenar algo, no exemplo abaixo com o Hover

```
.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}
```



```
.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
  &:hover {
    filter: drop-shadow(0 0 2em #646cffaa);
  }
  .react:hover {
    filter: drop-shadow(0 0 2em #61dafbaa);
  }
}
```


Usando SASS

Mixins

Mixins em SASS/SCSS são blocos de código reutilizáveis que podem ser incluídos em diferentes seletores. Eles são especialmente úteis para aplicar conjuntos de estilos repetitivos ou para lidar com prefixos específicos de navegadores. Se assemelham a funções que aplicam várias regras de CSS

```
@mixin border-radius($radius) {  
  -webkit-border-radius: $radius;  
  -moz-border-radius: $radius;  
  -ms-border-radius: $radius;  
  border-radius: $radius;  
}  
  
.button {  
  @include border-radius(5px);  
  background-color: #3498db;  
  color: white;  
  padding: 10px 15px;  
}
```

Usando SASS

Partials e Imports

Partials são arquivos SASS/SCSS que contêm partes de código reutilizáveis, como variáveis, mixins ou estilos específicos. Eles geralmente são nomeados com um prefixo _ (por exemplo, _variables.scss) e não são compilados diretamente em CSS. Em vez disso, eles são importados em outros arquivos SASS/SCSS usando a diretiva @import.

```
// _variables.scss
$primary-color: #3498db;
$secondary-color: #2ecc71;
$font-stack: 'Helvetica, Arial, sans-serif';
```

```
// main.scss
@import 'variables';
@import 'mixins';

.header {
  @include flex-center;
  background-color: $primary-color;
  color: white;
  height: 60px;
}
```

Usando SASS

Herança/Extensões

Herança, ou extensões, em SASS/SCSS permite que um seletor compartilhe estilos de outro seletor, evitando duplicação de código. Isso é feito usando a diretiva @extend. Esse recurso é útil para aplicar estilos comuns a múltiplos seletores, mantendo o código limpo e DRY (Don't Repeat Yourself).

```
.message {  
  padding: 10px;  
  border: 1px solid #ccc;  
  color: #333;  
}  
  
.success {  
  @extend .message;  
  border-color: green;  
}  
  
.error {  
  @extend .message;  
  border-color: red;  
}  
  
.warning {  
  @extend .message;  
  border-color: orange;  
}
```

Usando SASS

Condicionais (@if, @else, @else if)

No SASS/SCSS, você pode usar condicionais para aplicar estilos com base em certas condições. Isso é útil quando você precisa definir estilos diferentes dependendo de valores variáveis, como temas claros ou escuros, ou diferentes tamanhos de tela.

```
$theme: dark;

body {
  @if $theme == dark {
    background-color: #333;
    color: white;
  } @else {
    background-color: white;
    color: #333;
  }
}
```

Usando SASS

Loops (@for, @each, @while)

Loops no SASS/SCSS permitem iterar sobre listas ou realizar repetições baseadas em uma série de números. Eles são extremamente úteis para gerar conjuntos de estilos que seguem um padrão, como grades responsivas ou séries de classes utilitárias.

```
@for $i from 1 through 3 {  
  .col-#{ $i } {  
    width: (100% / 3) * $i;  
  }  
}
```


Dúvidas, críticas ou sugestões?

FIAP