eFront - Sass

Introdução

Assim como qualquer linguagem, sempre temos que evoluir e com CSS não é diferente! Neste pequeno guia iremos abordar o famoso Sass. Com ele, podemos criar variáveis, separar o nosso CSS em módulos, criar funções, implementar lógica de programação, e muito mais. Mas fique tranquilo, Sass e CSS são muito semelhantes, então se você tem a base de CSS a curva de aprendizado com Sass vai ser curta. Além disso, depois que você aprender a utilizar Sass em suas aplicações nunca mais vai voltar pro CSS puro.

O que é Sass

A sigla Sass significa “Syntactically Awesome Style Sheets” – ou seja, Folhas de Estilo com Sintaxe Espetacular. A sua ideia é adicionar recursos especiais ao escrever estilos como variáveis, mixins e importar grupos de estilos e outras opções que iremos comentar. Assim tornando o processo de desenvolvimento de estilos mais simples e eficiente do que com o CSS puro.

Iniciando com Sass

E como faço para iniciar com Sass, Iuri? É bem simples! Basta entrar em seu editor de código e criar um arquivo com o formato .scss exemplo: style.scss

Dessa forma estamos criando um documento Sass que irá criar estilos para o nosso documento HTML. Mas tenha calma... talvez você tenha tentando chamar o documento Sass no HTML mas não deu certo, por que disso?

Bem, estamos estilizando com Sass não quer dizer que o nosso documento Sass irá ser referenciado no HTML, mas sim o arquivo CSS. Confuso?

O que vai acontecer vai ser o seguinte, vamos desenvolver os estilos em Sass mas no final iremos ter um CSS normal referenciado no HTML.

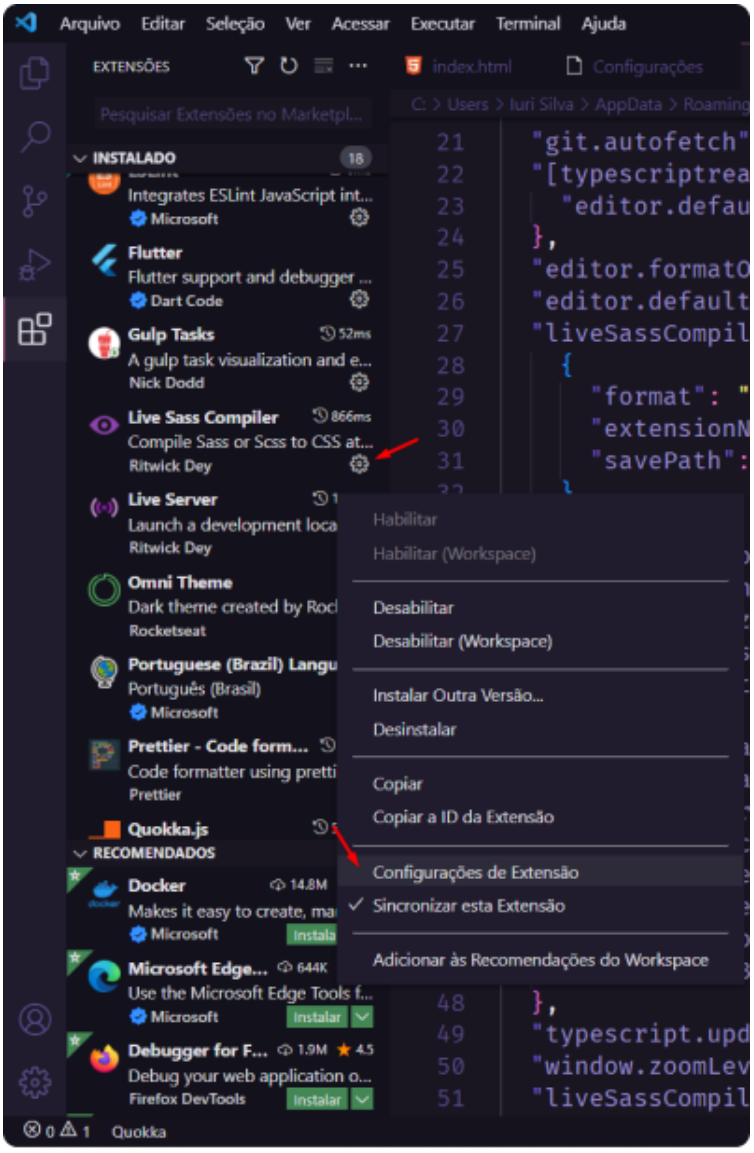
“Espera Iuri, quer dizer que todo nosso estilo com Sass vai virar um CSS no final?” Sim e calma, isso vai trazer muitas vantagens que iremos descobrir ao decorrer do material.

“Ok Iuri, mas como tenho acesso a esse CSS que foi feito pelo Sass?” Primeiro precisamos de uma ferramenta ou extensão para transpilar o documento Sass para CSS. Recomendo a extensão “Live Sass Compiler”.

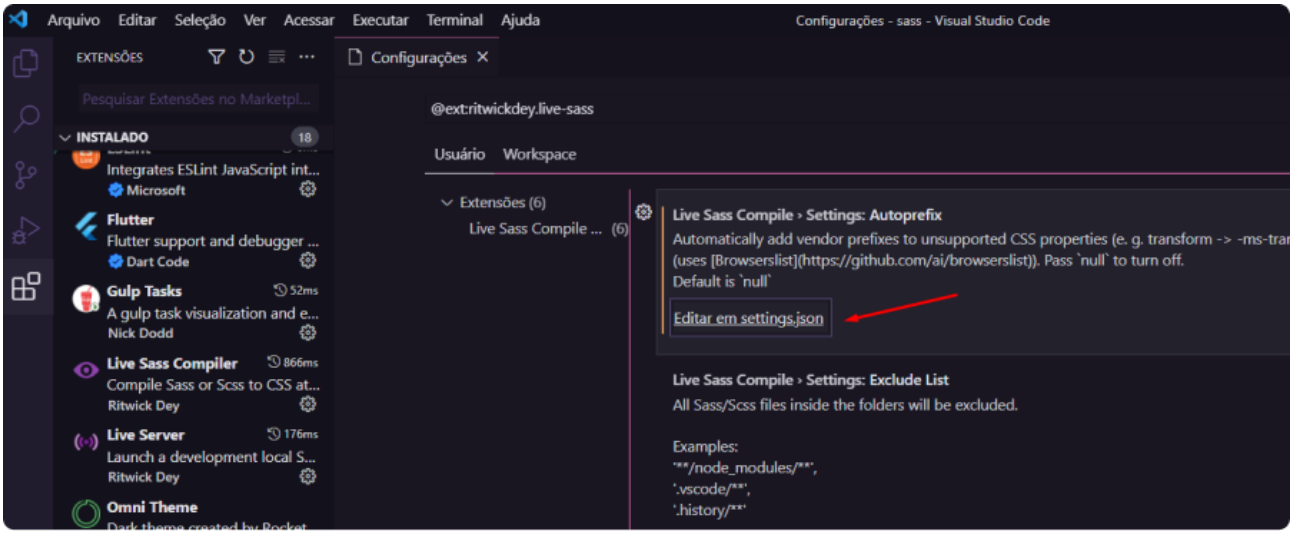
Ok, chega da teoria vamos para a prática.

Primeiro vou criar o diretório onde será “cuspido” o nosso CSS e assim referenciar ele no nosso HTML.

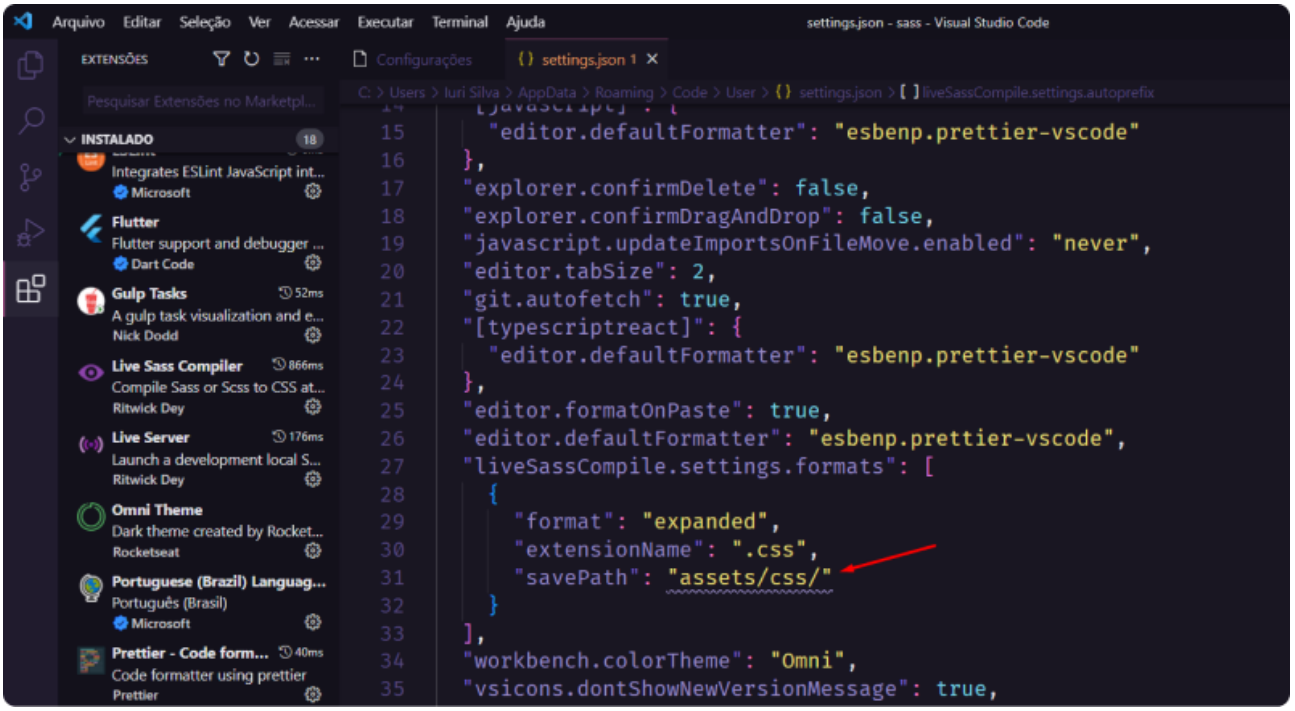
Depois de instalar a extensão “Live Sass Compiler” clique na engrenagem e vá em “Configurações de Extensão”.



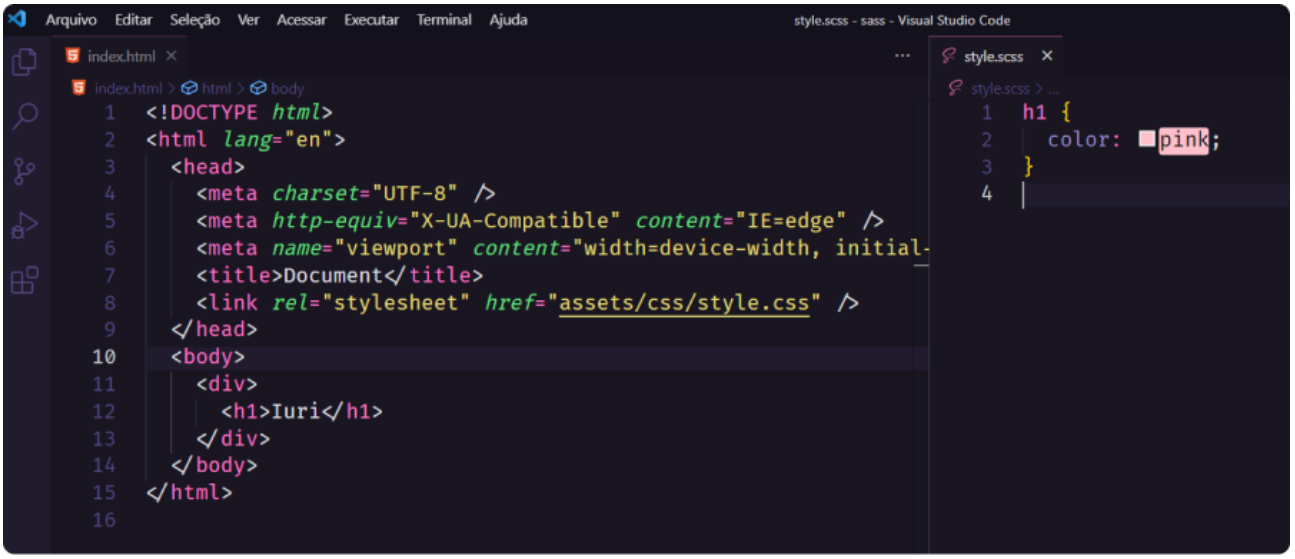
Depois clique em “Editar em settings.json”.



Na aba que se abre procure no “liveSassCompile” o “savePath” nele iremos colocar o caminho onde será transpilado o arquivo CSS. Veja que também temos outras duas opções, uma é o “extensionName” ele falaram qual será o formato que será transpilado, que em nosso caso é o .css e temos também a opção “format” que diz que se o CSS “cuspido” será extenso ou minificado.



Vamos fazer um teste para ver se nosso CSS está sendo transpilado corretamente. Para isso, crie um estilo qualquer e clique em “Watch Sass” localizado no barra inferior do editor de código (lembrando que essa opção será irá parecer se tiver utilizando a extensão “Live Sass Compiler”).
Ao fazer esse processo o próprio Sass deverá criar automaticamente o diretório, sendo assim precisamos só referenciar no nosso HTML o CSS transpilado pelo Sass.



Sintaxe

O Sass tem duas sintaxe diferentes, elas são: SCSS e SASS.

.scss

```scss
button {
  padding: 10px 20px;
  border-radius: 5px;
}
```

.sass

```sass
button
  padding: 10px 20px
  border-radius: 5px
```

SCSS: É bastante semelhante ao CSS. Você pode até dizer que SCSS é um superconjunto de CSS, o que significa que todo CSS válido também é SCSS válido. Devido à sua semelhança com CSS, é a sintaxe SASS mais fácil e popular usada.

SASS: Essa sintaxe tem todos os mesmos recursos do SCSS, a única diferença é que o SASS usa indentação em vez das chaves e ponto-e-vírgulas do SCSS.

Comentário

Os comentários no SCSS funcionam de maneira semelhante aos comentários em outras linguagens, como JavaScript. Comentários de linha única começam com // e vão até o final da linha, esses comentários não são mostrados no CSS compilado, somente comentários com /**/

```
// Este comentário não irá aparecer no CSS.

/* Mas este comentário irá, exceto no modo compactado. */
```

Variável

Podemos declarar uma variável utilizando o operador $, então passamos a propriedade. No Sass geralmente utilizamos variáveis para guardar um valor de cor, fonte, tamanho da fonte, largura ou altura, etc.

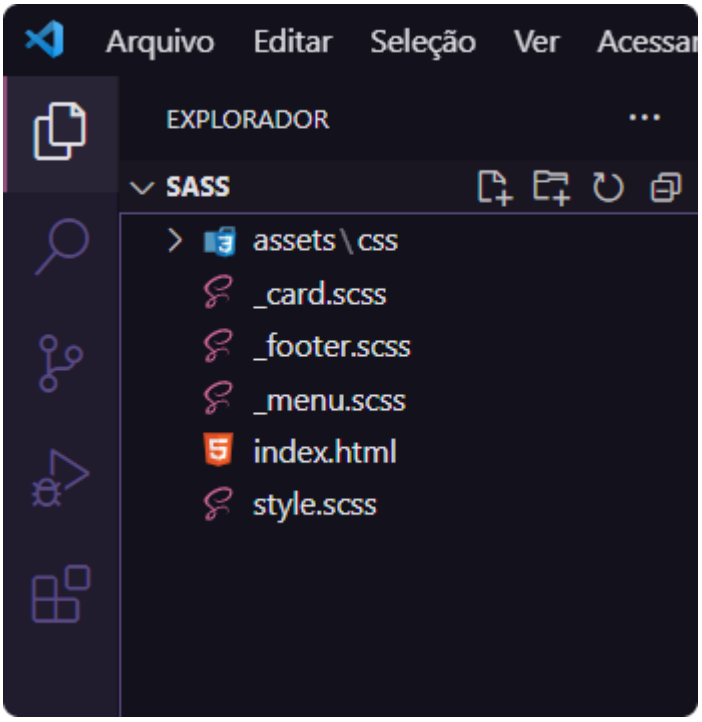
```
$cor: #0f0;

h1 {
  color: $cor;
}
```

Import/use

Muitas vezes queremos trabalhar em arquivos independentes (como componentes no React) pois, isso nos trás a vantagem de dar manutenção é só um documento em toda nossa aplicação. Isso é, se você tem um menu e um card no seu site, você pode separar os

estilos deles em parciais. Parciais são arquivos que são incluídos, mas não transpilam para arquivos CSS, isso é, podemos ter três arquivos .scss mas somente um será transpilado para um documento CSS.

Para incluir um parcial, você precisa seguir uma convenção de nomenclatura, que é colocar um underline no início do nome de cada arquivo. Ao fazer a importação, você não precisa colocar o underline (é opcional). Muitas vezes você vai ver esses arquivos parciais sendo importado com @import ou @use.




```
@import "menu.scss"; // Importando o _menu.scss no documento style.scss
```

Extend

O @extend permite compartilhar um conjunto de propriedades de um seletor para outro. Isso é muito útil se você tiver elementos de estilo quase idênticos que diferem apenas em alguns pequenos detalhes.

```
.button-simples {
 font-size: 16px;
 color: #0f0;
```

```
}

.button-vermelho {
 @extend .button-simples;
 background-color: red;
}
```

Esse é o resultado final do nosso CSS transpilado:

```
.button-simples, .button-vermelho {
 font-size: 16px;
 color: #0f0;
}

.button-vermelho {
 background-color: red;
}
```

Mixin

Algumas coisas em CSS são um pouco trabalhosa de escrever. Um mixin permite criar grupos de declarações CSS que você deseja reutilizar em todo o site. Você pode até passar valores para tornar seu mixin mais flexível.

```
@mixin theme {
 font-size: 14px;
 color: #fff;
}
```

```
.erro {
 @include theme;
 background-color: red;
}
```

Agora você deve estar se perguntando “Mas Iuri, isso é praticamente o extend”. De fato, se usarmos Mixins dessa forma não estamos aproveitando seu potencial.

Passando Variáveis para um Mixin

Mixins aceitam argumentos. Dessa forma, você pode passar variáveis para um mixin.

```
@mixin theme {
  background-color: $cor;
}

.erro {
  @include theme($cor: red);
}
```

No exemplo estamos dizendo que temos um mixin que espera receber um valor para o nosso background-color através do $cor.

Valor padrão para um Mixin

Também é possível definir valores padrão para variáveis mixin:

```
@mixin theme($cor: green) {
  background-color: $cor;
}

h1 {
  @include theme();
}

.erro {
  @include theme($cor: red);
}
```

No exemplo estamos dizendo que caso não seja definido o $cor no @include ele receba o valor padrão de “green”.

Nested

O Sass permitirá que você aninhe seus seletores CSS de uma maneira que siga a mesma hierarquia visual do seu HTML. Enquanto em CSS, as regras são definidas uma a uma (não aninhadas):

```scss
section {
   background-color: red;
   h2 {
    color: #000;
   }
}
```

.scss

```css
section {
  background-color: red;
}
section h2 {
  color: #000;
}
```

.css