React with Sass

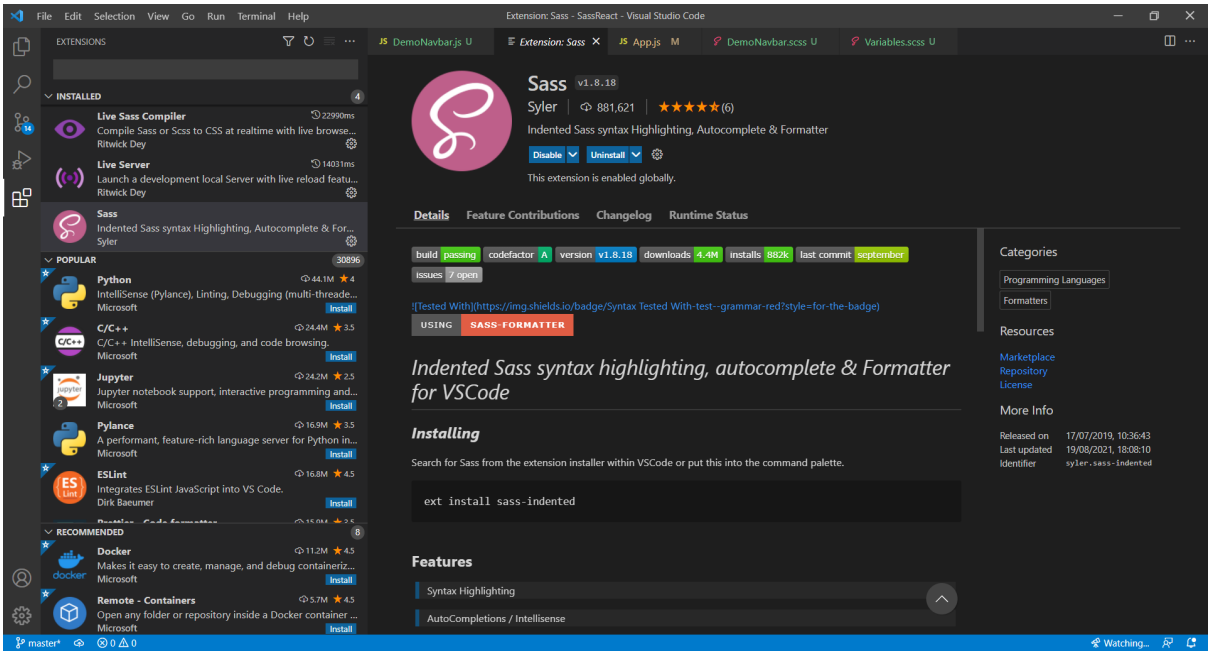
https://github.com/peerberger/SassReact

What is Sass?

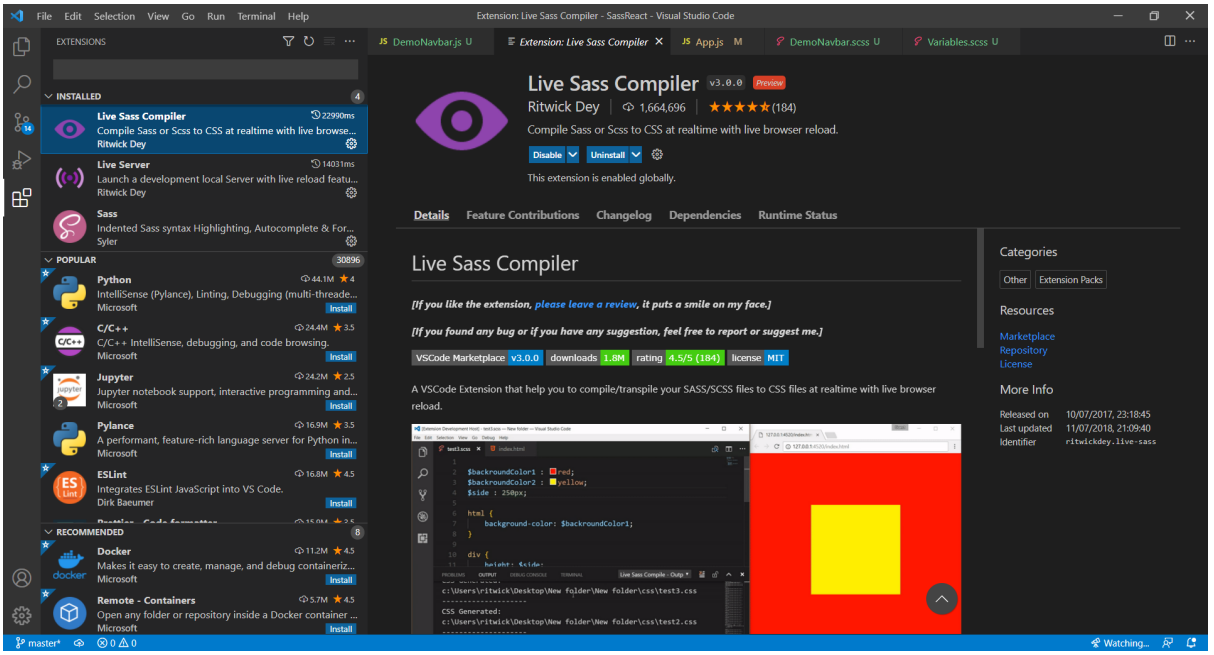
- Sass stands for **S**yntactically **A**wesome **S**tyle**s**heet.
- Sass is a more efficient way of writing css code.
- Sass reduces repetition of css code, which saves time.
- Sass lets you use features that do not exist in css, like:
    - Variables
    - Nested selectors
    - Mixins
    - Imports
    - Inheritance
    - Built-in functions
    - And more...

Useful VS Code Extensions

- Sass
    - Developed by Sass creators
    - Autocompletion
    - Formatting
    - Highlighting



- Live Sass Compiler
    - Essential
    - Automatically generates css files for sass code that you write yourself



Creating a React project

- Create a new React project with npm.
  Ex:

  npx create-react-app my-app

  Delete all files except **index.js**, **reportWebVitals.js**, and edit **App.js** so it looks like this (we will create the **DemoNavbar** component in the next step).
  Ex:

```
import DemoNavbar from "./Components/DemoNavbar";

function App() {
    return (
        <div className="App">
            <DemoNavbar></DemoNavbar>
        </div>
    );
}

export default App;
```

- Create a **Components** folder, and in it create a **DemoNavbar** component.
  Note that we're assigning to it a class named "**demoNavnar**", we will create it in the next section.
  Ex:

```jsx
import React from 'react';
import "../Styles/DemoNavbar.css";

function DemoNavbar() {
    return (
        <div className="demoNavbar">
            <a href="/home"> Home</a>
            <a href="/About"> About</a>
            <a href="/Profile"> Profile</a>
            <a href="/Contact"> Contact</a>
        </div>
    )
}

export default DemoNavbar;
```

Adding Sass to the project

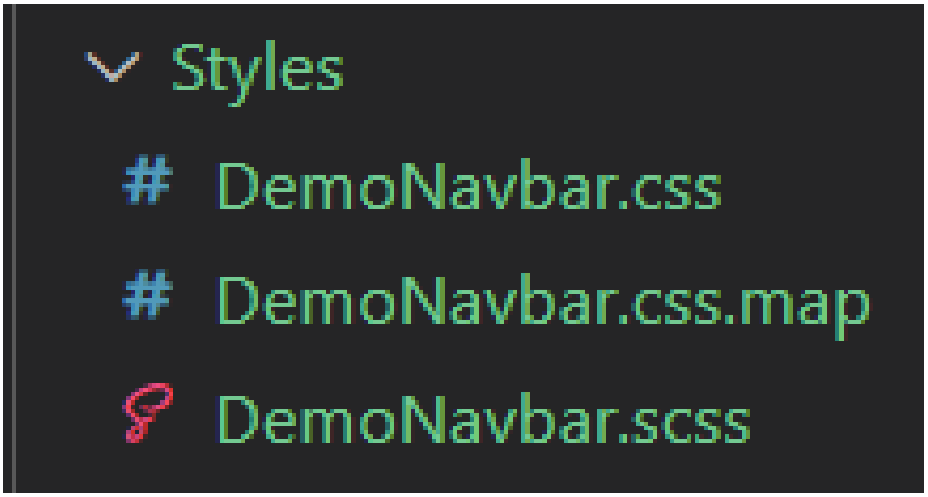
- **Basics**
  - Create a **Styles** folder, and in it create a **DemoNavbar** sass file (named **DemoNavbar.scss**).

  - In it, you can basically write plain css code, because there aren't a lot of differences between plain css and sass..
    Ex:

```scss
.demoNavbar {
    width: 100%;
    height: 80px;
    background-color: gray;
}
```

  - To make use of our **demoNavbar** class, we need to import it into the **DemoNavbar.js** file. But since the browser can't read sass files, we need to convert our sass code into css first.
  - To do that, make sure the **Live Sass Compiler** extension is working, by clicking on the "**Watch Sass**" tab on the bottom toolbar so it says "**Watching…**".
    Ex:

- Once you save the **DemoNavbar.scss** file, Live Sass Compiler will generate in the same folder 2 new files: **DemoNavbar.css**, and **DemoNavbar.css.map**.
  Ex:



  Right now the css code in **DemoNavbar.css** is exactly the same as **DemoNavbar.scss** of course, but once we start writing actual sass code, the extension will convert it to css in the new file.

- Now, to include the sass code we created in the **DemoNavbar** component, we need to import the generated **css** file, NOT the **scss** file (because you can't read sass files in the browser).
  Ex:

```
import React from 'react';
import "../Styles/DemoNavbar.css";

function DemoNavbar() {
    return (
        <div className="demoNavbar">
            <a href="/home"> Home</a>
            <a href="/About"> About</a>
            <a href="/Profile"> Profile</a>
            <a href="/Contact"> Contact</a>
        </div>
    )
}

export default DemoNavbar;
```

  If you run the app now, you should see the effects of the sass code.

- **Variables**
  - In sass you can create variables using the $ character, which can hold values like colors, fonts, etc.

  - To declare a sass variable, type a name for the variable, starting with $. Then, assign it a value just like normal css.
    Ex:

```
$blueColor: rgb(55, 165, 255);
```

  - To use a variable, type its name starting with $.
    Ex:

```
.demoNavbar {
    width: 100%;
    height: 80px;
    background-color: $blueColor;
}
```

  - A common practice in sass is to create separate files for variables, so you can use them wherever you want.
    So inside the **Styles** folder, create a **Variables** folder, and in it create a sass file called "**Variables.scss**", and simply move the variable definition line to there.
    Ex:

```
$blueColor: rgb(55, 165, 255);
```

  - Then import it in **DemoNavbar.scss**, but this time you **can** use the **scss** file. Also, make sure to type **@** before "**import**".
    Ex:

```
@import "../Styles/Variables/Variables.scss";

.demoNavbar {
    width: 100%;
    height: 80px;
    background-color: $blueColor;
}
```

  - Let's look at another variable example.
    Let's add the following variable for fonts.
    Ex:

    Variables.scss:

```
$fonts: Helvetica, Times, Serif;
```

Navbar.scss:

```
.demoNavbar a {
    font-family: $fonts;
}
```

Notice that if you use the **Live Server** extension to automatically update the app in the browser every time you save a file, you're gonna have to re-save **DemoNavbar.scss** for it to re-render, and re-generate the **DemoNavbar.css** file.

- **Nesting**
  - Normally, to target certain elements inside other elements, you'd use selectors like this.
    Ex:

```
.demoNavbar a {
    font-family: $fonts;
}
```

But with sass, you can organize them like the actual DOM.
Ex:

```
.demoNavbar {
    width: 100%;
    height: 80px;
    background-color: $blueColor;
    a {
        font-family: $fonts;
    }
}
```

- **Mixins**
  - Mixin are kind of like the functions of sass - if a variable contains one piece of data, mixins contain multiple pieces of data.
    So let's define a mixin in **Variables.scss**, and move some of the **demoNavbar** styles into it.
    Ex:

```
@mixin demoNavbarStyle {
    width: 100%;
    height: 80px;
    background-color: $blueColor;
}
```

Now, let's include this mixin back in **DemoNavbar.scss**.
Ex:

```
.demoNavbar {
    @include demoNavbarStyle();
    a {
        font-family: $fonts;
    }
}
```

- You can also pass parameters into mixins.
  Ex:

  Variables.scss:

```
@mixin demoNavbarStyle($width, $height, $backColor) {
    width: $width;
    height: $height;
    background-color: $backColor;
}
```

  Navbar.scss:

```
.demoNavbar {
    @include demoNavbarStyle(100%, 80px, $blueColor);
    a {
        font-family: $fonts;
    }
}
```

- **Inheritance**
  - Say you want to design a style class for a lot of elements, but you want some of them to be just a bit different.
    For example, you want all links to look the same, but the 'home' link to be **also** red.

  - You can define a class to assign to all links.
    Ex:

```
.linkBasic {
    padding: 15px 30px;
    font-size: 16px;
    cursor: pointer;
}
```

- And then define a class for just the ‘home’ link, that would “inherit” the styles already defined in the ‘basic’ class, using the **@extend** keyword.
  Ex:

```
.linkHome {
    @extend .linkBasic;
    color: red;
}
```

- Note that now you **don’t** need to assign **both** classes to the ‘home’ link, just the **linkHome** class.
  Ex:

```
function DemoNavbar() {
    return (
        <div className="demoNavbar">
            {/* <a href="/home" className="linkBasic
linkHome"> Home</a> */}
            <a href="/home" className="linkHome"> Home</a>
            <a href="/About" className="linkBasic">
About</a>
            <a href="/Profile" className="linkBasic">
Profile</a>
            <a href="/Contact" className="linkBasic">
Contact</a>
        </div>
    )
}
```

- **Built-in functions**
  - Sass has many built-in functions that are split into different categories. Here are some of each.

  - **String**
    - **quote(string)** - Adds quotes to string.
      Ex:

      quote(Hello) // “Hello”

    - **to-lower-case(string)** - Returns a copy of string in lower case.
      Ex:

      to-lower-case(Hello); // hello

- **Numeric**
  - **abs(number)** - Returns absolute value.
    Ex:

    abs(-15) // 15

  - **random(number)** - Returns a random integer between 1 and number.
    Ex:

    random(5) // 4

- **Introspection (useful for debugging)**
  - **call(function, arguments...)** - Calls a function with arguments, and returns the result.
    Ex:

    call(abs, -15) // 15

- **Color**
  - **lighten(color, amount)** - Lightens the color by the amount.
    Ex:

```
.linkHome {
    @extend .linkBasic;
    color: lighten($blueColor,10);
}
```