Conclusion et ressources supplémentaires

SASS

1. Références

- https://sass-lang.com/

2. Glossaire

- `SASS` = Syntactically Awesome Style Sheet (feuilles de style syntaxiquement impressionnantes)
- `Sassy` = insolent, culotté, impertinent
- `SCSS` = Syntactically Cascading Style Sheets

3. Préambule

- L'une des avancées les plus importantes de 2013 a été la maturation des processus d'outillage et de flux de travail pour les développeurs Web, y compris l'utilisation de préprocesseurs pour CSS et (dans une moindre mesure) HTML.
- Mais entre les raccourcis, les préprocesseurs, les frameworks et les postprocesseurs, il est facile de s'embrouiller dans le fouillis des technologies d'outillage.
- Malgré toute leur puissance, les navigateurs Web modernes ne comprennent que peu de choses : HTML, CSS, JAVASCRIPT et quelques formats multimédias. Ce sont les limites de ces formats qui génèrent le besoin de technologies d'assistance.

4. Introduction aux préprocesseurs

- Un préprocesseur est un logiciel qui permet de **transformer du code écrit dans une syntaxe spécifique en un autre code**, généralement dans un langage de programmation plus complexe ou plus proche de la machine.

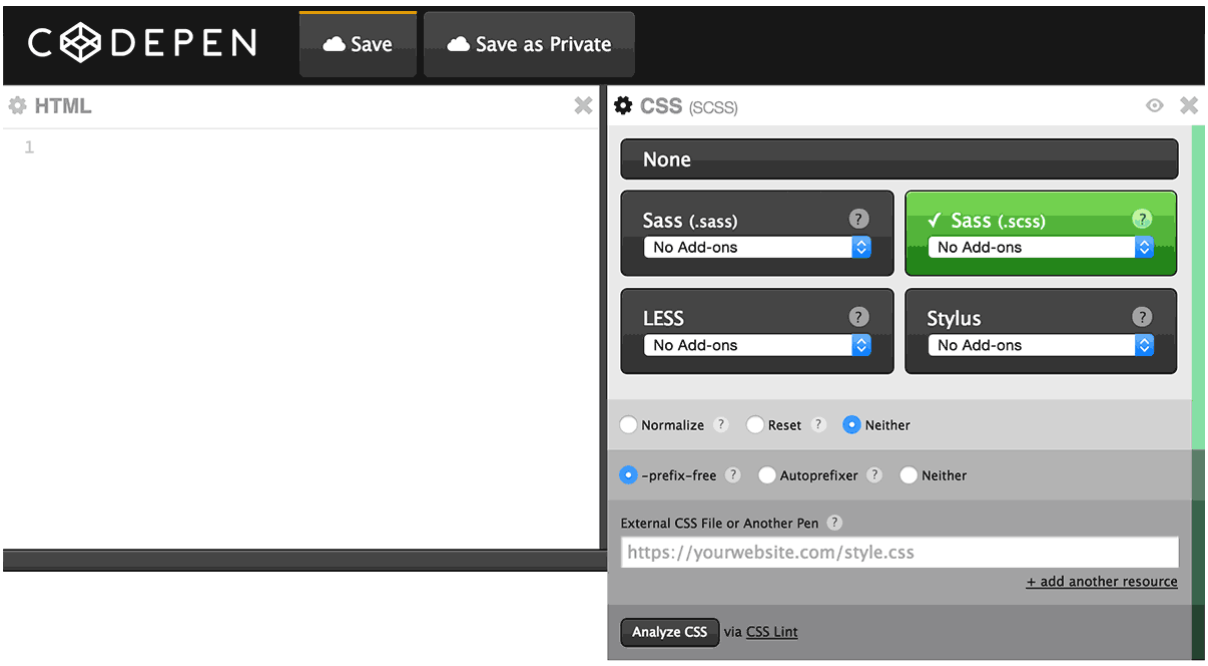
- Dans le contexte des feuilles de style CSS, un préprocesseur tel que `SASS` permet aux développeurs d'écrire du code CSS avec des fonctionnalités supplémentaires, telles que des **variables**, des **mixins** et des fonctions, qui ne sont pas disponibles dans le langage CSS standard.
- Le préprocesseur compile ensuite ce code en CSS valide pour les navigateurs web, ce qui permet aux développeurs d'écrire du code CSS plus efficacement et de manière plus modulaire.
- En somme, les préprocesseurs sont des outils qui simplifient le processus de développement en permettant aux développeurs d'écrire du code plus rapidement, plus efficacement et de manière plus maintenable.

5. SASS

> `SASS` (**Syntactically Awesome Style Sheets**) est un préprocesseur CSS qui permet aux développeurs d'écrire des feuilles de style CSS de manière plus efficace et élégante.

- SASS offre plusieurs fonctionnalités avancées, telles que les variables, les mixins, les fonctions, les boucles et les conditions, qui permettent aux développeurs de réutiliser du code et de créer des feuilles de style plus modulaires et maintenables.
- SASS permet également d'organiser les styles de manière plus logique et de les compiler en CSS valide pour les navigateurs web.
- En résumé, SASS est un outil puissant pour les développeurs front-end qui souhaitent écrire du code CSS plus rapidement, plus efficacement et de manière plus maintenable.
- Pour ceux qui souhaitent apprendre Sass, je suggérerais de suivre une voie qui évite entièrement tout investissement initial, n'a aucun engagement à long terme et demande très peu de temps : CodePen . De cette façon, vous pouvez commencer à coder Sass en seulement trois étapes simples :

1. Allez sur CodePen.io
2. Cliquez sur New Pen en haut à gauche
3. Sur l'écran qui apparaît, cliquez sur l'icône d'engrenage à côté de CSS et choisissez Sass (.scss)dans la nouvelle fenêtre. Cliquez à nouveau sur la même icône (ou n'importe où ailleurs sur l'écran) pour fermer la fenêtre.



Voilà : vous êtes maintenant prêt à commencer à coder avec Sass.

5.1. Deux choix de syntaxe SASS

Pourquoi deux choix de syntaxe Sass ?

> ### SCSS
>
> SCSS (Sassy CSS) est une version de Sass qui utilise une syntaxe plus proche de CSS standard. SCSS est donc plus facile à apprendre pour les développeurs qui sont familiers avec CSS, car elle utilise des accolades et des points-virgules pour délimiter le code, tout comme le CSS standard.

> ### SASS
>
> Sass (Syntactically Awesome Style Sheets) est la version originale de SASS, qui utilise une syntaxe indentée et concise pour écrire des feuilles de style. Sass a été développé en 2006 par Hampton Catlin et est largement utilisé par les développeurs front-end.

Les deux méthodes continuent d'être prises en charge, mais par souci de clarté (et pour faciliter la transition de CSS vers Sass), il est préférable d'utiliser la syntaxe `.scss`

SASS

```
$color: red
$color2: lime

a
  color: $color
  &:hover
    color: $color2
```

SCSS

```
$color: #f00;
$color2: #0f0;

a {
    color: $color;
    &:hover {
        color: $color2;
    }
}
```

CSS

```
a {
    color: red;
}
    a:hover {
        color: lime;
    }
```

6. Fonctionnalités

- variables
- nesting
- operateurs
- partial
- import
- mixin
- fonctions
- extends

6.1. Variables

> Un autre souci avec CSS c'est la répétition. Si par exemple vous avez une couleur accent sur votre site, cette couleur se répètera sans doute à plusieurs reprises dans votre code CSS. Si vous changez la couleur accent de votre image de marque et que vous voulez la changer partout sur votre site, vous devrez parcourir tous les fichiers et aller remplacer le code de couleur par le nouveau à chaque endroit où il est présent.

> Cette tâche peut être très longue, mais grâce à SASS vous avez maintenant une solution! Plutôt que d'écrire le code de votre couleur, vous allez créer une variable qui contiendra le code de votre couleur. De cette façon, si vous avez besoin de remplacer la couleur, vous la remplacer seulement dans la variable et la couleur sera changée partout sur votre site où il y a cette variable.

SCSS :

```
$vert: #468847;

p {
    color: $vert;
}
```

CSS :

```
p {
     color: #468847;
}
```

6.2. Commentaires

> Il existe deux types de commentaires dans Sass :
>
> - les commentaires sur une seule ligne : ils **ne seront pas** inclus dans le fichier CSS compilé.
> - les commentaires sur plusieurs lignes : ils **sont** inclus dans le fichier CSS compilé.

Commentaires sur une seule ligne :

SCSS :

```scss
// Ceci est un commentaire sur une seule ligne
$primary-color: #3498db;
div {
   background-color: $primary-color;
}
```

CSS :

```css
div {
   background-color: #3498db;
}
```

Commentaires sur plusieurs lignes :

SCSS :

```scss
/*
  Ceci est un commentaire
  sur plusieurs lignes
*/
$secondary-color: #f39c12;
```

CSS :

```css
/*
  Ceci est un commentaire
  sur plusieurs lignes
*/
.secondary-color {
  color: #f39c12;
}
```

6.3. Nesting

> SASS permet aux développeurs d'utiliser le `nesting`, c'est-à-dire la possibilité de définir des règles CSS imbriquées à l'intérieur d'autres règles CSS. Cette fonctionnalité permet de mieux organiser le code et de le rendre plus facile à lire et à comprendre.

SCSS :

```scss
$bleu: #2522da;
$vert: #468847;

ul {
   background-color: $bleu;
    li {
        color: $vert;
    }
}
```

CSS :

```css
ul {
     background-color: #2522da;
}

ul li {
     color: #468847;
}
```

6.4. Sélecteur parent `&`

> En Sass, le symbole `&` est utilisé pour faire référence au sélecteur parent dans un contexte de nesting.

Par exemple, si nous avons une règle CSS pour un lien (`a`) qui doit avoir un style différent lorsqu'il est survolé (`:hover`), nous pouvons utiliser `&` pour faire référence au sélecteur parent (`a`) dans le contexte du `:hover` :

SCSS :

```scss
a {
  color: blue;

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

CSS :

```
a {
    color: blue;
}

a:hover {
    text-decoration: underline;
}
```

Autre exemple

SCSS :

```
.card {
  background-color: #f1f1f1;
  padding: 20px;

  &__title {
    font-size: 24px;
    margin-bottom: 10px;
  }

  &__button {
    display: inline-block;
    padding: 10px 20px;
    border-radius: 5px;
    background-color: blue;
    color: white;

    &--yes {
      background-color: green;
    }

        &--yes {
      background-color: red;
    }
  }
}
```

CSS :

```
.card {
    background-color: #f1f1f1;
    padding: 20px;
}

.card__title {
    font-size: 24px;
    margin-bottom: 10px;
}
```

```
.card__button {
    display: inline-block;
    padding: 10px 20px;
    border-radius: 5px;
    background-color: blue;
    color: white;
}

.card__button--yes {
    background-color: green;
}

.card__button--yes {
    background-color: red;
}
```

6.5. Les extensions (@extends)

> L'instruction `@extend` en Sass permet de réutiliser les propriétés CSS d'une classe dans une autre classe.
>
> Cette fonctionnalité est utile pour éviter la duplication de code et pour maintenir un code CSS plus clair et plus modulaire.
>
> Afin d'alléger encore plus le code et d'éviter les répétitions, avec SASS il est possible de créer des blocs de propriétés et de les réutiliser à plusieurs endroits sans avoir besoin de les copier-coller.
>
> Vous écrivez le bloc de code une seule fois et ensuite l'insérez où vous voulez. Vos sélecteurs hériteront ainsi de déclarations communes à tous les sélecteurs.

SCSS :

```
// classe de base avec les propriétés CSS de base.
.btn {
  display: inline-block;
  padding: 10px 20px;
  border-radius: 5px;
  background-color: blue;
  color: white;
}

// classe qui hérite des propriétés de la classe de base
.btn--large {
  @extend .btn;
  font-size: 20px;
  padding: 15px 30px;
}
```

CSS :

```css
.btn, .btn--large {
     display: inline-block;
     padding: 10px 20px;
     border-radius: 5px;
     background-color: blue;
     color: white;
}


 .btn--large {
     font-size: 20px;
     padding: 15px 30px;
}
```

6.6. Les Mixins

> Les `mixins` (morceaux de code CSS réutilisables.) permettent de créer des blocs de code que vous pouvez ensuite inclure dans d'autres règles.
>
> « Oui, mais c'est la même chose que les extension non?! »... et bien non :
>
> - Avec les `mixins` il est possible d'ajouter des variables en plus.
> - L'ajout des variables n'est pas possible avec les extensions.
> - Imaginez que vous avez plusieurs blocs avec des coins arrondis sur votre site, mais avec des angles différents, vous pouvez créer une `mixin` et mettre une variable pour gérer les arrondis de vos blocs et ainsi éviter les copier-coller dans votre code.

SCSS :

```scss
$radius: 10px;


@mixin border-radius($radius) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
      -ms-border-radius: $radius;
          border-radius: $radius;
}


.box1 { @include border-radius($radius); }
.box2 { @include border-radius(15px); }
```

CSS :

```
.box {
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  -ms-border-radius: 10px;
  border-radius: 10px;
}
```

6.7. Placeholder %

> Les placeholders en Sass sont des sélecteurs CSS qui ne génèrent pas de code CSS dans le fichier de sortie CSS.
>
> Les placeholders sont définis à l'aide de l'instruction `%` et sont généralement utilisés pour définir des styles réutilisables pour des éléments HTML spécifiques.

SCSS :

```
%link-style {
  text-decoration: none;
  color: blue;

  &:hover {
    text-decoration: underline;
  }
}

a {
  @extend %link-style;
}

.bt2n {
  @extend %link-style;
  display: inline-block;
  padding: 10px 20px;
  border-radius: 5px;
  background-color: blue;
  color: white;
}
```

CSS :

```
a, .btn2 {
     text-decoration: none;
     color: blue;
}
 a:hover, .btn2:hover {
     text-decoration: underline;
}
```

```
.btn2 {
    display: inline-block;
    padding: 10px 20px;
    border-radius: 5px;
    background-color: blue;
    color: white;
}
```

6.8. @extends vs Mixins vs Placeholders

> En Sass, les `placeholders`, les `mixins` et les `@extends` sont des outils différents pour gérer le code CSS réutilisable. Bien qu'ils puissent sembler similaires à première vue, ils ont des fonctions différentes et sont utilisés dans des contextes différents.

1. Les placeholders (`%`) sont des sélecteurs CSS qui ne génèrent pas de code CSS dans le fichier de sortie CSS. Ils sont utilisés pour définir des styles réutilisables pour des éléments HTML spécifiques, et sont étendus à l'aide de l'instruction `@extend`. Les placeholders sont utiles pour réduire la duplication de code et pour définir des styles qui ne sont pas liés à une classe spécifique.
2. Les mixins (`@mixin`) sont des blocs de code Sass qui peuvent être inclus dans d'autres blocs de code à l'aide de l'instruction `@include`. Les mixins sont utilisés pour encapsuler des styles réutilisables qui peuvent accepter des arguments et générer différents résultats en fonction des paramètres fournis. Les mixins sont utiles pour définir des styles génériques qui peuvent être adaptés à différents contextes.
3. Les `@extends` sont des instructions Sass qui permettent d'étendre les styles d'un sélecteur CSS existant à un autre sélecteur. Les `@extends` sont utiles pour éviter la duplication de code en héritant des styles existants et pour générer des styles plus clairs et plus modulaires. Cependant, l'utilisation excessive des `@extends` peut entraîner une complexité et une confusion accrues, en particulier lors de la maintenance du code.

A RETENIR :

- les `placeholders` sont utilisés pour définir des styles réutilisables pour des éléments HTML spécifiques.
- les `mixins` sont utilisés pour encapsuler des styles réutilisables génériques.
- les `@extends` sont utilisés pour hériter des styles existants et éviter la duplication de code.
- il est important de comprendre les différences entre ces outils et de les utiliser judicieusement en fonction des besoins spécifiques du projet

6.9. Les partiels (partials) et les importations (@import)

> En Sass, les fichiers dont le nom commence par `_` sont des fichiers de partials.

> Les fichiers de partials sont destinés à être inclus dans d'autres fichiers Sass à l'aide de l'instruction `@import`.
>
> Lorsque Sass compile le code CSS, il ne génère pas de fichier CSS pour les fichiers de partials, car ils sont destinés à être inclus dans d'autres fichiers Sass.

_variables.scss :

```
$primary-color: #007bff;
$secondary-color: #6c757d;
```

style.scss :

```
@import 'variables';

body {
  background-color: $primary-color;
}

.header {
  background-color: $secondary-color;
  color: white;
}
```

style.css :

```
body {
     background-color: #007bff;
}

.header {
     background-color: #6c757d;
     color: white;
}
```

6.10. Les fonctions intégrées

> Sass propose un ensemble de fonctions intégrées pour effectuer des opérations mathématiques, des manipulations de chaînes de caractères, des opérations de couleurs et bien plus encore:

SCSS :

```
$color1: #007bff;
$color2: #6c757d;
$color3: #dc3545;
```

```scss
$color4: #28a745;

.alert {
  background-color: rgba($color3, 0.8);
  color: $color4;
  border: 1px solid darken($color3, 10%);
}

.btn-primary {
  background-color: $color1;
  color: white;

  &:hover {
    background-color: lighten($color1, 10%);
  }

  &:active {
    background-color: darken($color1, 10%);
  }
}

.btn-secondary {
  background-color: $color2;
  color: white;

  &:hover {
    background-color: lighten($color2, 10%);
  }

  &:active {
    background-color: darken($color2, 10%);
  }
}

.btn-success {
  background-color: $color4;
  color: white;

  &:hover {
    background-color: lighten($color4, 10%);
  }

  &:active {
    background-color: darken($color4, 10%);
  }
}

.btn-danger {
  background-color: $color3;
  color: white;

  &:hover {
```

```
    background-color: lighten($color3, 10%);
  }

  &:active {
    background-color: darken($color3, 10%);
  }
}
```

CSS :

```
.alert {
     background-color: rgba(220, 53, 69, 0.8);
     color: #28a745;
     border: 1px solid #bd2130;
}
 .btn-primary {
     background-color: #007bff;
     color: white;
}
 .btn-primary:hover {
     background-color: #3395ff;
}
 .btn-primary:active {
     background-color: #0062cc;
}
 .btn-secondary {
     background-color: #6c757d;
     color: white;
}
 .btn-secondary:hover {
     background-color: #868e96;
}
 .btn-secondary:active {
     background-color: #545b62;
}
 .btn-success {
     background-color: #28a745;
     color: white;
}
 .btn-success:hover {
     background-color: #34ce57;
}
 .btn-success:active {
     background-color: #1e7e34;
}
 .btn-danger {
     background-color: #dc3545;
     color: white;
}
 .btn-danger:hover {
     background-color: #e4606d;
```

```
}
 .btn-danger:active {
     background-color: #bd2130;
}
```

7. Compiler SASS localement

Sass doit être installé sur votre système avant de pouvoir l'utiliser pour compiler des fichiers Sass en CSS à partir de la ligne de commande.

Il y a deux façons d'installer Sass :

1. Utiliser `npm` (Node Package Manager) : Sass peut être installé en utilisant la commande `npm` depuis votre terminal ou ligne de commande. Voici la commande à utiliser :

```
npm install -g sass
```

Cette commande installe Sass globalement sur votre système.

2. Télécharger le binaire : Sass peut également être téléchargé et installé manuellement en téléchargeant le binaire depuis le site officiel de Sass.

Une fois que Sass est installé sur votre système, vous pouvez utiliser la commande `sass` à partir de votre terminal ou ligne de commande pour compiler des fichiers Sass en CSS.

Pour compiler des fichiers Sass en CSS à partir de la ligne de commande, vous pouvez utiliser la commande `sass` fournie par Sass.

Voici les étapes à suivre pour compiler un fichier Sass en CSS :

1. Ouvrez un terminal ou une ligne de commande sur votre système d'exploitation.
2. Accédez au dossier contenant votre fichier Sass à l'aide de la commande `cd`.
3. Utilisez la commande `sass` pour compiler votre fichier Sass en CSS. Par exemple, si votre fichier Sass s'appelle `style.scss` et que vous voulez compiler en CSS dans un fichier `style.css`, vous pouvez utiliser la commande suivante :

```
sass style.scss style.css
```

Cette commande indique à Sass de compiler le fichier `style.scss` en CSS et de le sauvegarder dans le fichier `style.css`.

Si vous souhaitez surveiller le fichier Sass pour les modifications et le recompiler automatiquement lorsque des modifications sont apportées, vous pouvez utiliser l'option `--watch`.

Par exemple :

```
sass --watch style.scss:style.css
```

Cette commande indique à Sass de surveiller le fichier `style.scss` pour les modifications et de recompiler automatiquement en CSS dans le fichier `style.css`.

Sass est un outil puissant pour générer du code CSS dynamique et réutilisable. En utilisant la ligne de commande pour compiler des fichiers Sass en CSS, vous pouvez facilement intégrer Sass dans votre flux de travail de développement et générer un code CSS efficace pour vos projets.

8. OUTILS SASS / SCSS

8.1. Compilateur en ligne

- https://jsonformatter.org/scss-to-css
- https://beautifytools.com/scss-compiler.php

9. Extension VSCode : Live Sass Compiler

> Sources :
>
> - https://ritwickdey.github.io/vscode-live-sass-compiler/docs/settings.html

```
"liveSassCompile.settings.formats":[
  // This is Default.
  {
      "format": "expanded",
      "extensionName": ".css",
      "savePath": null
  },
  // You can add more
  {
      "format": "compressed",
      "extensionName": ".min.css",
      "savePath": "/dist/css"
  },
```

```
  // More Complex
  {
      "format": "compressed",
      "extensionName": ".min.css",
      "savePath": "~/../css"
  }
]
```

10. SASS & BOOTSTRAP

10.1. Installation de Bootstrap et SASS :

- Pour utiliser Bootstrap avec SASS, vous devez d'abord installer les deux packages via npm (Node Package Manager).
- Ouvrez votre terminal ou invite de commande et exécutez les commandes suivantes :

```
npm init (pour créer un fichier package.json si vous n'en avez pas déjà un)
npm install bootstrap
npm install sass
```

10.2. Importer Bootstrap dans votre fichier SASS :

- Créez un fichier SASS (par exemple, `styles.scss`) dans votre projet et importez Bootstrap en ajoutant l'instruction `@import` en haut du fichier.

```
@import '~bootstrap/scss/bootstrap';
```

10.3. Personnalisation de Bootstrap :

Vous pouvez personnaliser Bootstrap en modifiant ses variables SASS avant d'importer le fichier `bootstrap.scss`. Par exemple, pour changer la couleur principale et la police de caractères, ajoutez les lignes suivantes avant l'instruction `@import` :

```
$primary: #FF5722;
$font-family-base: 'Roboto', sans-serif;


@import '~bootstrap/scss/bootstrap';
```

La commande `~bootstrap/scss/bootstrap` est une déclaration `@import` en Sass qui permet d'importer les fichiers SCSS de la bibliothèque Bootstrap dans un fichier SCSS.

Plus précisément, le symbole tilde `~` est utilisé pour indiquer à Sass de chercher le dossier `bootstrap` à partir de la racine du projet. Dans ce cas, Sass cherchera le dossier `bootstrap` dans le dossier `node_modules` installé à la racine du projet.

La suite du chemin `scss/bootstrap` indique à Sass de chercher les fichiers SCSS de Bootstrap dans le dossier `scss` situé dans le dossier `bootstrap`.

En utilisant cette commande, vous pouvez inclure les fichiers SCSS de Bootstrap dans votre propre fichier SCSS et les personnaliser selon les besoins de votre projet. Lorsque vous compilez le fichier SCSS, les styles de Bootstrap sont compilés avec vos styles personnalisés pour créer un fichier CSS final qui peut être inclus dans votre page HTML.

Plus précisément, le dossier `bootstrap` contient des fichiers SCSS qui définissent les styles de la bibliothèque Bootstrap. Les fichiers SCSS sont utilisés pour créer les fichiers CSS finaux qui sont inclus dans une page HTML. En important les fichiers SCSS de Bootstrap dans votre propre fichier SCSS, vous pouvez personnaliser les styles de Bootstrap et les intégrer à vos propres styles pour créer une apparence unique pour votre site ou application.

Cependant, il n'est pas nécessaire d'inclure l'extension `.scss` dans la déclaration `@import`. En effet, Sass est capable de déterminer automatiquement le type de fichier à importer en fonction de l'extension du fichier importé.

Lorsque vous importez les fichiers SCSS de Bootstrap dans votre propre fichier SCSS, vous pouvez choisir d'importer l'ensemble des fichiers SCSS de Bootstrap ou seulement les fichiers dont vous avez besoin pour votre projet.

Si vous souhaitez importer tous les fichiers SCSS de Bootstrap, vous pouvez utiliser la déclaration `@import '~bootstrap/scss/bootstrap';` dans votre fichier SCSS principal. Cette déclaration importera tous les fichiers SCSS de Bootstrap, y compris les fichiers de base, les composants, les utilitaires, les variables, etc.

Si vous ne souhaitez importer que certains fichiers SCSS de Bootstrap, vous pouvez spécifier les fichiers que vous voulez importer à la place de `~bootstrap/scss/bootstrap` dans votre déclaration `@import`. Par exemple, pour importer uniquement les fichiers de base de Bootstrap, vous pouvez utiliser la déclaration `@import '~bootstrap/scss/bootstrap-reboot';`.

Voici quelques exemples de fichiers SCSS que vous pouvez importer selon vos besoins:

- `bootstrap.scss`: inclut tous les fichiers SCSS de Bootstrap, y compris les fichiers de base, les composants, les utilitaires, les variables, etc.
- `bootstrap-reboot.scss`: inclut les fichiers de base de Bootstrap, y compris les réinitialisations CSS et les styles globaux.
- `bootstrap-grid.scss`: inclut les fichiers SCSS pour la grille Bootstrap.
- `bootstrap-buttons.scss`: inclut les fichiers SCSS pour les boutons Bootstrap.
- `bootstrap-forms.scss`: inclut les fichiers SCSS pour les formulaires Bootstrap.
- `bootstrap-card.scss`: inclut les fichiers SCSS pour les cartes Bootstrap.
- `bootstrap-modal.scss`: inclut les fichiers SCSS pour les modaux Bootstrap.

Notez que si vous utilisez un système de gestion de tâches tel que Gulp ou Webpack, vous pouvez également inclure les fichiers SCSS requis dans votre configuration de compilation Sass pour compiler uniquement les fichiers dont vous avez besoin.

10.4. Compiler le fichier SASS en CSS :

Après avoir apporté les modifications souhaitées à votre fichier SASS, vous devez compiler le fichier en CSS pour que les navigateurs puissent l'interpréter. Vous pouvez le faire en utilisant le package `sass` que vous avez précédemment installé via npm. Dans votre terminal ou invite de commande, exécutez la commande suivante :

```
npx sass styles.scss styles.css
```

Cette commande prendra votre fichier `styles.scss`, y compris les modifications apportées à Bootstrap, et le compilera en un fichier CSS (`styles.css`) que vous pourrez lier à vos pages HTML.

10.5. Lier le fichier CSS à votre HTML :

Enfin, ajoutez une balise `<link>` dans l'en-tête de votre fichier HTML pour inclure le fichier CSS compilé :

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Mon site avec Bootstrap et SASS</title>
  <link rel="stylesheet" href="./css/styles.css">
</head>
<body>
  <!-- Contenu de votre site ici -->
</body>
</html>
```

10.6. Fichier .map

Le fichier `.map` est un fichier de mappage source-source qui est généré lors de la compilation de fichiers Sass ou CSS. Ce fichier permet de cartographier les fichiers Sass ou CSS compilés aux fichiers sources Sass ou CSS d'origine.

Le fichier `.map` contient des informations telles que le nom du fichier d'origine, les numéros de ligne et de colonne correspondants, ainsi que les informations de mappage pour les règles CSS compilées. Ces informations sont utilisées pour aider les développeurs à déboguer leur code en fournissant une correspondance entre les styles CSS générés et le code source Sass ou CSS d'origine.

Le fichier `.map` est souvent utilisé en conjonction avec les outils de développement tels que les inspecteurs de navigateur pour faciliter le processus de débogage. Les navigateurs modernes prennent en charge la lecture des fichiers de mappage source-source, ce qui permet aux développeurs de voir le code source Sass ou CSS d'origine dans les outils de développement, même s'il est compilé en CSS.

Notez que la génération de fichiers de mappage peut ralentir le processus de compilation, donc si vous n'avez pas besoin de cette fonctionnalité, vous pouvez désactiver la génération de fichiers de mappage en passant l'option `--no-source-map` lors de la compilation de vos fichiers Sass ou CSS.