← 8 astuces pour optimiser votre Sass

Lorsqu'il est utilisé efficacement, Sass permet de créer des feuilles de style en cascade (CSS) évolutives et saines. Cependant, lorsqu'il est mal utilisé, Sass peut augmenter la taille des fichiers et ajouter du code inutile ou en double.

Voici une série de conseils et d'astuces pour vous aider à tirer le meilleur parti de Sass.

1. STRUCTUREZ VOTRE SASS

Pour tout nouveau projet Sass, il est essentiel que la structure de votre site soit correcte dès le départ. L'utilisation de fichiers partiels vous permet de diviser le CSS en blocs de code plus petits et plus faciles à gérer, qui sont plus faciles à maintenir et à développer.

Les fichiers partiels sont créés à l'aide d'un trait de soulignement et ne sont pas édités sous forme de fichiers CSS distincts. Chaque fichier partiel doit être importé à l'aide d'un fichier Sass “général”(global.scss, main.scss, master.scss, style.scss... à vous de choisir le nom) à la racine du dossier Sass.

Voici par exemple une structure de fichier assez classique qui correspond à cette pratique :

```
sass/
|
|– base/
|   |– _reset.scss       # Reset/normalize
|   |– _typography.scss  # Typography rules
|   ...                  # Etc…
|
|– components/
|   |– _buttons.scss     # Buttons
    # Etc…
|
|– layout/
|   |– _navigation.scss  # Navigation
|   |– _grid.scss        # Grid system
|   ...                  # Etc…
|
|– pages/
|   |– _home.scss        # Home specific styles
|   |– _contact.scss     # Contact specific styles
|   ...                  # Etc…
|
|– sass-utils/
|   |– _variables.scss   # Sass Variables
|   |– _functions.scss   # Sass Functions
|   |– _mixins.scss      # Sass Mixins
|   |– _helpers.scss     # Class & placeholders helpers
|
|– vendors/
|   |– _bootstrap.scss   # Bootstrap
|   ...                  # Etc…
|
|
`– main.scss            # Primary Sass file
```

Cette structure de dossiers garantit que le site est facile à enrichir. Par exemple, de nouveaux modules peuvent facilement être ajoutés au dossier module, puis ajoutés à `global.scss` à l'aide de `@import`.

2 - UTILISEZ LES VARIABLES SASS PLUS EFFICACEMENT

Les variables sont l'une des fonctions les plus simples à comprendre de Sass, mais elles sont encore parfois mal utilisées. Il est essentiel de créer une convention de

nommage pour l'ensemble du site lorsque vous travaillez avec des variables. Sans cela, elles deviennent plus difficiles à comprendre, à réutiliser et à maintenir, ce qui réduit fortement leur intérêt.

Voici quelques conseils pour créer des variables utiles :

- Ne soyez pas trop vague lorsque vous nommez vos variables.
- Ayez et respectez une convention de nommage (Modular, BEM, etc.).
- Assurez-vous que l'utilisation de la variable est justifiée

Voici quelques bons exemples :

```
$orange: #ffa600;
$grey: #f3f3f3;
$blue: #82d2e5;

$link-primary: $orange;
$link-secondary: $blue;
$link-tertiary: $grey;

$radius-button: 5px;
$radius-tab: 5px;
```

Et quelques mauvais exemples :

```
$link: #ffa600;
$listStyle: none;
$radius: 5px;
```

3 - RÉDUISEZ L'UTILISATION DES MIXINS

Les mixins sont un excellent moyen d'inclure des sections de code plusieurs fois sans avoir besoin de les réécrire. Cependant, inclure un mixin revient à copier et coller les styles dans tout le fichier CSS une fois qu'il sera compilé, ce qui crée une masse de code dupliqué et peut gonfler votre fichier CSS.

Un mixin ne doit donc être utilisé que si vous l'exploitez via des arguments, pour créer

rapidement des styles modifiés.

Par exemple :

```
@mixin rounded-corner($arc) {
    -moz-border-radius: $arc;
    -webkit-border-radius: $arc;
    border-radius: $arc;
}
```

Cette mixin `rounded-corner` peut être utilisée dans toutes les situations qui le nécessitent et s'adapte aux différents cas en changeant la valeur de `$arc`

```
.tab-button {
     @include rounded-corner(5px);
}

.cta-button {
     @include rounded-corner(8px);
}
```

Et un mauvais exemple ressemblerait à :

```
@mixin cta-button {
    padding: 10px;
    color: #fff;
    background-color: red;
    font-size: 14px;
    width: 150px;
    margin: 5px 0;
    text-align: center;
    display: block;
}
```

Cette mixin n'a pas d'argument, ce serait bien plus efficace dans ce cas d'utiliser un placeholder.

4. ADOPTEZ. LES. PLACEHOLDERS.

A contrario des mixins, les placeholders peuvent être utilisés et réutilisés sans générer de duplication de code à la compilation, ce qui en fait une option bien plus efficace lorsque l'on essaye de produire du DRY (Don't Repeat Yourself) CSS.

Ce code SCSS :

```
%bg-image {
    width: 100%;
    background-position: center center;
    background-size: cover;
    background-repeat: no-repeat;
}

.image-one {
    @extend %bg-image;
    background-image:url(/img/image-one.jpg");
}

.image-two {
    @extend %bg-image;
    background-image:url(/img/image-two.jpg");
}
```

Sera compilé en ce code CSS :

```
.image-one, .image-two {
    width: 100%;
    background-position: center center;
    background-size: cover;
    background-repeat: no-repeat;
}

.image-one {
    background-image:url(/img/image-one.jpg") ;
}

.image-two {
    background-image:url(/img/image-two.jpg") ;
}
```

Le code répété dans le placehodler n'est présent qu'une seule fois et seuls les styles uniques sont appliqués aux sélecteurs individuels. S'ils ne sont pas utilisés, les styles du placeholder ne sont pas du tout compilés dans le CSS final.

En lien avec le point précédent, les placeholders peuvent être utilisés avec les mixins pour réduire le code répété tout en gardant la flexibilité d'un mixin...

```
/* PLACEHOLDER
============================================= */

%btn {
    padding: 10px;
    color:#fff;
    curser: pointer;
    border: none;
    shadow: none;
    font-size: 14px;
    width: 150px;
    margin: 5px 0;
    text-align: center;
    display: block;
}

/* BUTTON MIXIN
============================================= */

@mixin  btn-background($btn-background) {
    @extend %btn;
    background-color: $btn-background;
    &:hover {
        background-color: lighten($btn-background,10%);
    }
}

/* BUTTONS
============================================= */

.cta-btn {
    @include btn-background(green);
}

.main-btn {
    @include btn-background(orange);
}

.info-btn {
    @include btn-background(blue);
}
```

5. UTILISEZ DES FONCTIONS POUR VOS CALCULS

Les fonctions sont utilisées pour effectuer des calculs. Une fonction Sass ne produit

aucun fichier CSS. Au lieu de cela, elle renvoie une valeur qui peut être utilisée dans le CSS. C'est utile pour les calculs qui seront effectués dans l'ensemble du site.

Par exemple, les fonctions sont utiles pour calculer le pourcentage de largeur d'un élément donné :

```
@function calculate-width ($col-span) {
    @return 100% / $col-span
}

.span-two {
    width: calculate-width(2); // spans 2 columns, width =
}

.span-three {
    width: calculate-width(3); // spans 3 columns, width =
}
```

6. ORGANISEZ VOTRE TRAVAIL

Placez tous les mixins, les fonctions, les placeholders et les variables dans un fichier partiel dédié. En gardant ces blocs de code ensemble, vous vous assurez qu'ils sont faciles à modifier et à réutiliser à l'avenir.

Les éléments globaux du site doivent être regroupés dans un dossier de base. Le dossier de base doit contenir le fichier de variables globales telles que les polices et les schémas de couleurs :

```
$font-primary: 'Roboto', sans-serif;
$font-secondary: Arial, Helvetica, sans-serif;

$color-primary: $orange;
$color-secondary: $blue;
$color-tertiary: $grey;
```

Les mixins, fonctions et variables spécifiques à un module doivent être conservés dans le fichier contenant les fichiers SCSS partiel lié au module concerné.

7. LIMITEZ L'IMBRICATION

L'utilisation excessive de règles imbriquées dans vos fichiers SCSS peut entraîner de nombreux problèmes, allant d'un code complexe à maintenir à une spécificité trop excessive, en passant par une trop grande dépendance à l'égard de la structure HTML d'une page. Ces éléments peuvent causer des problèmes plus tard et vous forcer à surpasser la hiérarchie des règles via l'usage du keyword `!important` par exemple.

Voici quelques règles d'or pour l'imbrication :

- Ne jamais dépasser 3 niveaux de profondeur.
- Assurez-vous que le résultat CSS est propre et réutilisable.
- Utilisez l'imbrication quand elle est utile, et non comme une option par défaut.

8. VISEZ LA SIMPLICITÉ

En conclusion, et ce conseil s'applique à de nombreux domaines du développement, visez la simplicité finale du résultat. L'objectif de Sass est de permettre aux développeurs d'écrire du code CSS plus propre et plus facile à maintenir et à faire évoluer.

Donc avant de créer une nouvelle mixin, d'ajouter des variables pour tout et n'importe quoi, ou de prévoir des fonctions pour des calculs dont vous n'aurez jamais vraiment besoin, assurez-vous que leur présence améliore le développement. Chaque ajout augmente la complexité du code mais certains sont plus pertinents que d'autres.

Créer une liste de variables sans fin, sans utilisation claire, ou des fonctions complexes qui sont difficiles à comprendre pour toute autre personne que celui qui les a mises en place est contre-productif et n'aboutira pas à la production d'un code CSS propre, léger et performant.

CONCLUSION

C’est tout pour cette petite liste de conseils au sujet de Sass. La technologie Web est en constante évolution et les pratiques changent en fonction des nouvelles possibilités.