front-end developer_
/webdesigner

BEM Methodology

Block Element Modifier

Bruxelles Formation | CEPEGRA | Olivier Céréssia | 2017

webdesigner.cepegra-labs.be

PRÉAMBULE

Quand on écrit du code CSS, à l'heure actuelle, chacun le fun un peu « à sa sauce », avec sa propre logique qui lui est propre.

Si cette façon de fonctionner vous convient tout à fait et vous permettra probablement de vous y retrouver de prime abord, il est fort possible que, vous-même, sur des projets que vous avez codés, serez un peu « perdus » lorsqu'il s'agira, quelques années ou mois plus tard, de mettre à jour ce site avec de nouveaux impératifs demandés par votre équipe ou votre client.

Que dire alors des gens qui vont devoir reprendre votre code CSS en main avant de se l'approprier et finalement de le modifier ? Ou de vous-même si vous devez débarquer dans une entreprise où vous devez prendre en main le travail d'un autre front-end dev. pcq celui-ci ne respectait pas la méthodologie mise en place par l'équipe de développement ?

Pour rendre notre code plus facile à lire et à re-lire, des méthodologies d'écriture de nos classes et d'identification des éléments qui composent nos pages ont été mises en place, nous pouvons en citer quelques-unes :

- OOCSS
- SMACSS
- BEM
- Etc.

Toutes ces approches sont là pour une seule raison : accroître votre lisibilité, la structure de vos fichiers CSS et votre rapidité à produire du code. Nous allons nous attarder sur une des méthodes les plus couramment employées sur le Web : la méthode BEM.

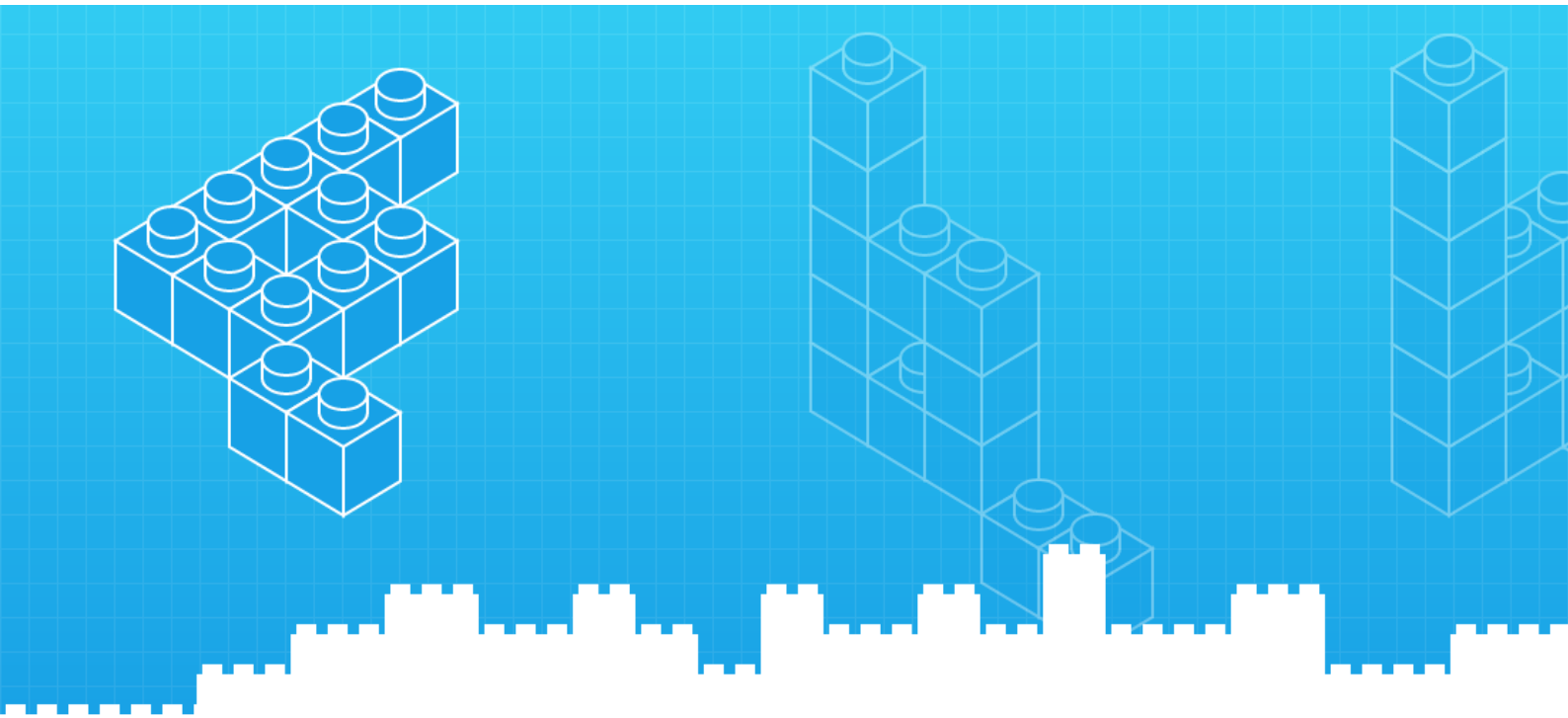


TABLE DES MATIÈRES

BEM – BLOCK ELEMENT MODIFIER

• Présentation

BEM est l'abréviation de Block Element Modifier.

Cela suggère une façon structurée de nommer vos classes CSS, basé sur les propriétés de l'élément en question. Avec la méthodologie BEM, vous allez retrouver des classes qui auront ce genre de structure : `header__form-email` par exemple.

Quand on utilise la méthodologie BEM, on n'utilise évidemment pas les ID mais bien des classes (comme toujours). Parce que grâce au fait que ce sont des classes, on va pouvoir les répéter si c'est nécessaire et on va aussi pouvoir créer une structure de code (HTML / CSS et SASS) plus consistante.

• Block

Le *Block* est le conteneur ou le contexte où les éléments se retrouvent eux-mêmes.

Ils définissent ce sur quoi ils sont placés et non ce à quoi va ressembler ce sur quoi ils sont placés.

Quand vous pensez au Block, vous devez les identifier comme étant les plus gros éléments structurels de votre code.

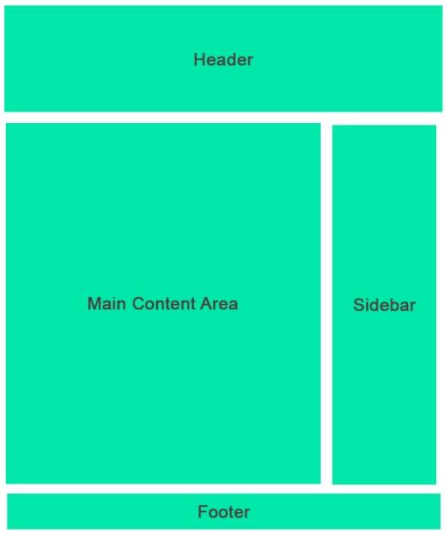
On pourrait y retrouver des balises comme header, footer, sidebar ou main ; chacune de ses balises serait considérée comme un élément Block (dans la logique BEM).

En regardant l'image ci-à-côté, on a une structure tout à fait classique d'une page Web.

Les éléments Block sont donc au nombre de 4 ici.

Les éléments de type Block, dans la méthodologie BEM sont ceux qui vont commencer vos classes :

```
.header_etc
.main_etc
.sidebar_etc
.footer_etc
```



Dès que vous avez défini vos éléments Block, vous serez prêts à commencer à nommer vos éléments.

Les Blocks peuvent être imbriqués les uns dans les autres et vous pouvez avoir un nombre infini d'imbrication de Blocks les uns dans les autres.

• Element

L'Element est une partie d'un Block qui ne peut pas être utilisé sans un élément de type Block.

Comme pour les éléments Block, l'Element permet de définir de quoi il s'agit et pas ce à quoi ça va ressembler. C'est pour ajouter de la précision au sein d'un Block.

Un Element est une partie optionnelle d'un Block, tous les Blocks n'ont donc pas forcément un Element.

Le Block est l'ensemble et les Elements sont les parties du Block. Chaque Element est écrit après le Block et ils sont connectés par deux underscore de cette manière :

```
.block__element
```

Voilà donc à quoi va ressembler une déclinaison d'un Element au sein d'un Block.

On pourrait donc par exemple avoir, au sein d'un header :

```
.header__logo { … }
.header__tagline { … }
.header__searchbar { … }
.header__navigation { … }
```

Vous gardez donc encore la main sur le nom des différents Elements, mais il est probable que les Block, eux, soient souvent les mêmes. Vous gardez donc une certaine créativité et surtout vous gardez la main sur le nom que vous allez donner à l'Element.

Par exemple, navigation pourrait devenir nav, tagline pourrait devenir tag ou tagLine. Ce que vous devez garder à l'esprit, c'est que tout ceci doit rester le plus simple, clair et précis possible.

Les Elements sont le cœur du nom de vos classes, c'est grâce aux Elements que vous allez pouvoir structurer vos feuilles de style.

• Quand choisir entre un Block ou un Element ?

1. Si une portion de code doit pouvoir être réutilisée et qu'elle ne dépend pas d'autres composants qui doivent être implémentés sur la page, vous devriez créer un Block.
2. Si la portion de code ne peut être utilisée séparément sans son entité parente (le Block), un Element est généralement créé.

Il existe une exception pour des éléments qui doivent être scindés en plus petites parties (sous-elements) pour faciliter le développement. Dans la méthodologie BEM, vous ne pouvez pas créer des éléments d'éléments. Dans un cas comme celui-ci, plutôt que de créer un élément, vous avez besoin de créer un Block.

- # Modifier

Le Modifier est l'entité qui définit l'apparence, l'état ou le comportement d'un Block ou d'un Element.

Le nom du Modifier est séparé d'un block ou d'un élément par un simple underscore (_).

Le Modifier décrit donc :

- L'apparence (Quelle taille `size_s` ou `size_xl` par exemple)
- Son état (En quoi est-il différent des autres `disabled` , `focused`, etc.)
- Son comportement (Qu'est-il sensé faire ? `directions_left-top`)

Quand vous nommez une classe, l'intention est d'aider à faire en sorte que cet Element soit répétable et qu'on puisse le réutiliser dans d'autres parties du site sans avoir à le répéter alors que les styles sont les mêmes.

Quand vous devez modifier le style d'un Element spécifique, vous pouvez utiliser un Modifier. Pour ce faire, dans la méthodologie BEM, vous allez ajouter un double - - juste après l'Element ou le Block.

Par exemple :

```
.block--modifier { … }
.block__element--modifier { … }
```

Attention, c'est ici qu'il faut être vigilant pour ne pas créer des classes inutiles et garder votre code CSS le plus propre et simple possible, sans répétition.

Partons d'un exemple concret :

```
.header__navigation { … }
.header__navigation--secondary { … }
```

Un header qui compterait deux navigations en son sein dont une secondaire. Il y a pas mal de chances pour que les navigations partagent certains styles mais qu'ils aient également certaines différences. Avec Sass, vous pouvez @extend la navigation principale comme ça la secondaire va avoir les mêmes propriétés et ensuite lui ajouter ses particularités comme ceci :

```
.header__navigation {
   background : #008cba;
   padding : 1rem 0 ;
   text-transform : uppercase ;
 }
.header__navigation--secondary {
   @extend .header__navigation ;
   background : #dfe0e0 ;
 }
```

Avec BEM, vous n'avez à utiliser qu'une classe par balise HTML. Par exemples pour les labels :

```
.label { … }
.label--alert { … }
```

Des langages comme SASS permettent

rapidement de partager les mêmes styles et en acceptant de petites exceptions (avec les @extend par exemple).

- ## Mix

On peut évidemment utiliser différentes entités de la méthodologie BEM au sein d'un seul élément du DOM.

Les Mix vont vous permettre de :

- Combiner le comportement et les styles de différentes entités sans dupliquer du code
- Créer des éléments de l'interface sémantiquement neufs mais basés sur des éléments qui sont déjà existants

Par exemple

```
<!-- `header` block -->
<div class="header">
    <!--
        The `search-form` block is mixed with the `search-form` element
        from the `header` block
    -->
    <div class="search-form header__search-form"></div>
</div>
```

Dans cet exemple, on a combiné le comportement et les styles du Block `search-form` et de l'élément `search-form` qui descend du Block header.

Cette approche nous permet de mettre tout ce qui concerne la géométrie et le positionnement de la boite sur l'élément `header__search-form` alors que le Block `search-form` restera universal et réutilisable en dehors de n'importe quel contexte.