CSS / JS Manifesto

Bonnes pratiques et conventions

Tables des matières :

Structure des fichiers

CSS

Tous les fichiers de style doivent se trouver dans le répertoire `public/css/`. Ce répertoire est subdivisé en deux répertoires : `lib/` et `layout/`.

Note : *Structure du répertoire dans le cas où chaque site est une application différente.*

```
/public/
/public/css/
/public/css/lib/
/public/css/lib/{lib-file}.css
/public/css/layout/{module}/
/public/css/layout/{module}/{controller}.css
/public/css/layout/blocks/{name}.css
/public/css/layout/common.css
```

Répertoire lib/

Dans ce répertoire doivent se trouver :

- Les fichiers CSS liés à des libraries JS.
  Exemple: jquery-ui.css (les styles par défaut des plugins jQuery UI).
  Note : Toujours préfixer le nom du fichier par le nom de la librarie.
  Exemple : *jquery*-modal.css
- Le fichier common.css, rassemblant les styles communs à l'ensemble de l'application.

Répertoire layout/

Dans ce répertoire doivent se trouver les fichiers de styles propres aux différentes pages du site, organisés par modules et controllers, ainsi que les fichiers de styles des différents blocs, dans le répertoire `blocks`.

Nommage des fichiers de template :
`{module}/{controller}.css`

JS

Tous les fichiers JS doivent se trouver dans le répertoire `public/js/`. Ce répertoire est subdivisé en deux répertoires : `lib/` et `layout/`.

```
/public/
/public/js/
/public/js/lib/
/public/js/lib/{lib-file}.js
/public/js/layout/{module}/
/public/js/layout/{module}/{controller}.js
```

Répertoire lib/

Dans ce répertoire doivent se trouver :

- Les libraries et plugins JS, qu'elles soient créées par nous ou non.
  Exemple: jquery.js, xt_core.js, ou encore jquery-modal.js

  Note : S'il s'agit d'un plugin lié à une librarie JS, toujours préfixer le nom du fichier par le nom de la librarie, et suffixer par la version. `{parent-lib}-{name}-{version}.js`
  Exemple : *jquery*-modal-1.2.js

Répertoire layout/

Dans ce répertoire doivent se trouver :

- les fichiers JS propres aux différentes pages du site, organisés par modules et controllers, ainsi que les fichiers JS des différents blocs, dans le répertoire `blocks`.
  Nommage des fichiers de template :
  `{module}/{controller}.js`
- Le fichier common.js, rassemblant les fonctions ou les évènements communs à l'ensemble de l'application.

Images

A définir.

Helpers

Objectif : mettre au point un helper JS et un helper CSS capables d'inclure automatiquement les fichiers JS et CSS de base dans une action :

- Les fichiers communs à toutes les pages (common, xiti...)
- Le fichier propre à l'action en cours (module/controller)

Piste :
http://leakingabstraction.com/2010/02/01/managing-css-and-javascript-files-within-a-zend-framework-app/

Conventions de nommage et de codage

CSS

Nommage

Les noms des classes et des IDs doivent être tout en minuscule, toujours commencer par une lettre (jamais de chiffres au début), et le séparateur est l'underscore ou le tiret (pas de camelcase).

Commentaires

Intitulez chacune de vos sections par un bloc de commentaire :

```
/*************************************************
 * Header
*************************************************/
```

Si vous avez d'ajouter des sous-sections, utilisez :

```
/* Logo */
```

Quand utiliser une classe, et quand utiliser un ID ?

Rappels : La principale différence entre les IDs et les classes est que les IDs doivent être UNIQUES sur une page, alors que les classes peuvent se retrouver plusieurs fois.

Il n'y a pas de règle officielle quant à l'utilisation des classes ou des IDs, mais voici la préconisation :

- **.classes :** Utilisez des classes lorsque vous voulez définir un style réutilisable sur d'autres éléments.
  Par exemple : .bold, .title, .box, .button
- **#id :** Utilisez des IDs quand l'élément que vous avez à définir est un élément ou une section de la page. Celui-ci soit être clairement identifié sur la page.
  Par exemple : #menu, #pagination, #footer, #sidebar
  Note : Si l'élément en question est clairement identifiable dans la page, mais qu'il est présent plusieurs fois, utilisez des classes.

Typiquement, les styles définis par des .classes auront plus leur place dans des fichiers de styles situés dans le répertoire lib/, alors que styles définis par des #id auront plus leurs place

dans des fichiers situés dans le répertoire layout/.
Exemple concret :
Si vous devez styliser la description d'une annonce en gris clair + gras, préférez affectez les classes .bold et .grey à votre div, plutôt que de créer une classe .desc et de définir son style à l'intérieur.

Attention au code inutile

Voici quelques exemples très basiques mais démontrant le propos :

Exemple 1 :
Dans cet exemple, inutile d'attribuer un ID à <ul>.

Pas bon :

```
<div id="menu">
	<ul id="nav">
	</ul>
</div>

#nav {
	margin: 10px
}
```

Bon :

```
<div id="menu">
	<ul>
	</ul>
</div>

#menu ul {
	margin: 10px
}
```

Exemple 2 :
Dans cet exemple, la couleur des liens ne changeant pas selon l'état, elle peut être définie en même temps pour les deux. Quant au soulignage, il sera pris en compte au hover puisque **a:hover** est plus précisément défini que **a**.

Pas Bon :

```
#menu ul li a {
	color: blue;
	text-decoration: none;
}

#menu ul li a:hover {
	color: blue;
	text-decoration: none;
}
```

Bon :

```
#menu ul li a,
#menu ul li a:hover {
	color: blue;
	text-decoration: none;
}

#menu ul li a:hover {
	text-decoration: underline;
}
```

Préciser le plus possible vos règles de style

Afin d'éviter les conflits et de devoir ajouter des !important partout, préciser au maximum les éléments parents de vos styles. En effet, les règles de styles prioritaires sont celles les plus précisément définies.

Exemple :
#header #menu ul li a { color: blue; } ← **Bien !**
#menu a { color: blue; } ← **Ne sera pas pris en compte car surchargé par la règle ci-dessus**

Web Sémantique & HTML5

Attention à utiliser les bonnes balises. Par exemple, si vous avez un bloc de texte, encadrez le de <p> et non de . Idem pour les titres, utilisez des <h1>, <h2>, etc, plutôt que de simples .

Avec HTML5, de nouvelles balises sémantiques sont disponibles. N'hésitez pas à les utiliser, elles sont compatibles tous navigateurs.

<section>
Utilisez <section> plutôt que pour identifier les sections principales d'une page.
Exemple : un bloc annonce dans une PR.

<nav>
Section contenant des liens de navigation.
Exemple : le menu sur toutes les pages, ou bien les liens suivant / précédent en FA, ou encore les liens dans le footer.

<aside>
Des éléments qui sont reliés au contenu mais sans en faire partie.
Exemple : les blocs dans la colonne de droite en PR et FA.

<header>
Section faisant office d'en-tête d'une page ou d'une section.
Exemples : Header sur toutes les pages, titres des blocs sur la HP

En savoir plus :
http://tcuvelier.developpez.com/tutoriels/web-semantique/html5-microdonnees/introduction/

Compatibilité entre les navigateurs

Il arrive (très) souvent que certains navigateurs nécessitent des règles CSS à part pour afficher le site correctement. Pour pallier à ce problème, plusieurs solutions :

- Inclure des fichiers CSS spécifiques selon le navigateur
- Utiliser des hacks
- Ajouter des classes au body selon le navigateur via un plugin jQuery.

C'est la dernière solution qui est préconisée. Par exemple, les classes "ie", et "ie7" seront affectées au <body> si le site est affiché sous IE7. Il suffira ensuite de préfixer nos règles CSS propre à IE7 :

```
#menu ul li a {
	padding: 10px;
}

.ie7 #menu ul li a {
	padding: 11px;
}
```

De cette façon, toutes les règles CSS liées à un élément restent rassemblées au même endroit.

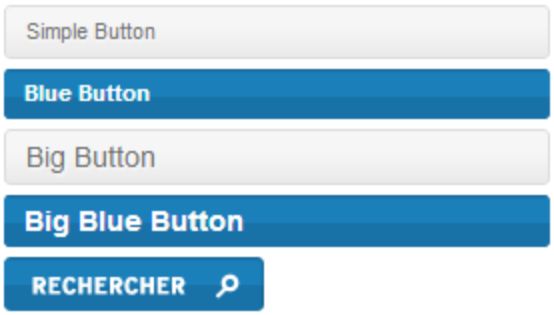
En PHP ?
http://blog.pastel.pro/appliquer-des-attributs-css-seulement-a-un-navigateur-et-une-version/

CSS Helpers

Afin de garantir une certaine cohérence dans la charte graphique et une meilleure maintenabilité du code, nous devons organiser nos classes CSS sous forme de helpers.

Plus concrètement, les divers éléments de l'interface (boutons, liens, pictos, champs de formulaires...) ont généralement un style définit par la charte graphique, commun à l'ensemble du site. Il faut donc définir ces styles dans les fichiers CSS commun, et **toujours** les utiliser.

Exemple des boutons :



Si pour un nouveau projet vous avez besoin d'un bouton dont le style n'existe pas déjà dans les fichiers communs, alors ajoutez-le en surchargeant la classe .btn existante.

Prenons une classe .btn :

```
.btn {
	background: #ccc;
	font-size: 11px;
	border: 1px solid #ccc;
}
```

Si vous avez besoin d'un bouton ayant une police de 14px, créez la classe suivante :

```
.btn-big {
	font-size: 14px;
}
```

Et construisez votre bouton de la façon suivante :

```
<a class=”btn btn-big”>bouton</a>
```

JS

Hébergement en local V.S Hébergement CDN pour jQuery

Le principal avantage dans l’utilisation d’un CDN pour servir jQuery est que tous les visiteurs ayant déjà téléchargé le fichier via ce CDN n’auront pas à le retélécharger en arrivant sur LICOM.

Le CDN le plus répandu étant celui de Google, la préco est ce dernier, même si ce n’est pas tout à fait le plus rapide.

OnLoad

Afin que votre code soit exécuté lorsque la page a terminé son chargement, vous devez le placer dans une fonction anonyme déclenchée par l’évènement onLoad. Il existe plusieurs façon de le faire, mais voici celle préconisée :

```
(function($) {
        // Votre code
})(jQuery);
```

Cette méthode a un avantage : celui de prévenir les conflit. En effet, l’objet jQuery est passé en paramètre, et affecté à $. De cette façon, à l’intérieur de cette fonction anonyme, $ fait référence à jQuery, sans risque de conflit avec d’autres libraries.

Plugins jQuery

Afin de nous forcer à correctement penser et structurer nos codes JS, notre objectif va être de créer des plugins jQuery ou utiliser des plugins existants.

Exemple :

Un besoin fréquent est de faire en sorte que les champs texte des formulaires se vident au clic. Nous pouvons développer un plugin jQuery qui fait exactement cela. Nous allons lui donner comme nom : **autoClear**.

Déclarer le plugin

Créons le fichier du plugin dans `/public/js/lib/jquery-autoclear-1.0.js`

Déclarer un plugin revient en fait à ajouter une méthode à la bibliothèque jQuery. Pour cela, il faut utiliser la fonction **fn()** de jQuery :

```
(function($) {
        $.fn.autoClear = function() { ... };
})(jQuery);
```

autoClear() est désormais une méthode jQuery que nous pouvons appeller sur n'importe quel élément du DOM.

Exemple :
$('input.text').autoClear();

DocBlock

```
/* ------------------------------------------------------------------------
      Class: autoClear
      Use: Clears input fields when clicked
      Author: Vincent Ballut
      Version: 1.0
      Dependency: jQuery 1.7
------------------------------------------------------------------------ */
```

Passer des paramètres à la fonction

Il existe deux façons de passer des paramètres à une fonction :

- En arguments
  `$('input.text').autoClear(0, 100);`
  Mais si nous décidons d'ajouter d'autres paramètres plus tard ? Notre plug-in pourrait avoir des dizaines d'options — la gestion des paramètres devient rapidement compliquée.
- Sous forme de tableau JSON
  $('input.text').autoClear({ param1: 0, param2: 100 });

Avec la deuxième méthode, nous devons gérer les paramètres reçu, et pourquoi pas déclarer des valeurs par défaut. Ceci se fait via la méthode extend() de jQuery :

```
(function($) {
	$.fn.autoClear = function(params) {
		params = $.extend( {param1: 0, param2: 50}, params);
	};
})(jQuery);
```

Retour de la fonction

jQuery utilise le chaînage extensivement des méthodes, nous devons donc penser à toujours retourner l'objet this à la fin de toutes nos méthodes

```
(function($) {
	$.fn.autoClear = function(params) {
		params = $.extend( {param1: 0, param2: 50}, params);
		// Traitement...
		return this;
	};
})(jQuery);
```

Tuto Vidéo ultra complet sur jQuery : https://tutsplus.com/course/30-days-to-learn-jquery/

Gestion des évènements

Afin de conserver tout le code JS au même endroit, attention à bien définir les évènements dans les fichiers JS et non dans le code HTML :

`<a onclick="doSomething()">` ← **Beurk**

`$("a").on('click', function() { doSomething(); });` ← **Bien :)**