WordPress

La création de thèmes et le développement d'extensions

Olivier Dommange
2017

Sommaire

WordPress est un CMS destiné à priori à la création de blogs. Ses fonctionnalités et la palette de modules qu'offre ce CMS en font un outil de référence dans la mise à jour de contenus pour un site Internet. Son usage s'est donc élargi à de nouvelles vocations qui demeurent pour le moins modestes mais qui permet d'envisager WordPress comme une solution dans la création de plusieurs sites Internet.

L'administration d'un site avec WordPress est plutôt simple grâce notamment à une interface agréable et facile d'utilisation et proposant les fonctionnalités essentielles à la mise à jour de contenus et la gestion des utilisateurs.

Développer un thème dédié à WordPress

Il est question de créer un thème propre aux exigences visuelles et graphiques qui ont été définies dans une maquette conçue à l'aide d'un logiciel de traitement d'images à partir de laquelle une page en HTML et CSS a été générée.

L'administration des contenus d'un thème s'établit par l'insertion de fonctions WordPress en PHP. Ces fonctions permettent de rendre dynamique le contenu qui pourra être créé, modifié ou supprimé par les administrateurs du site.

Les fonctionnalités des thèmes ne s'arrêtent pourtant pas là. Il est ainsi possible d'intégrer des fonctionnalités plus complexes :

- Créer de multiples menus assignés à des positions définies dans le thème.
- Gérer des zones réservés aux widgets.
- Personnaliser l'administration des articles afin que ceux-ci disposent de champs supplémentaires correspondant aux besoins du site avec l'extension « Advanced Custom Fields ».
- Créer des sous-pages destinés à des pages spécifiques telles que la page d'accueil.
- Rendre le thème administrable et permettre la personnalisation du thème.

L'installation du CMS

L'emploi de WordPress implique d'utiliser un serveur Web. Dans un cadre de développement, il est utile d'utiliser un serveur Web local tel que EasyPHP (www.easyphp.org), Mamp (www.mamp.info), Wamp (wampserver.com) Xampp (https://www.apachefriends.org) ou Lamp (pas besoin de l'installer il est automatiquement disponible sur les systèmes d'exploitation linux) qui disposent :

- du logiciel HTTP Apache
- de la base de données MySQL
- de la librairie PHP
- de l'outil de gestion des bases de données PhpMyAdmin

Ces outils sont ceux dont dispose un hébergeur Web. Ce qui signifie qu'un transfert des fichiers du CMS et de la base de données permet la mise en ligne du site.

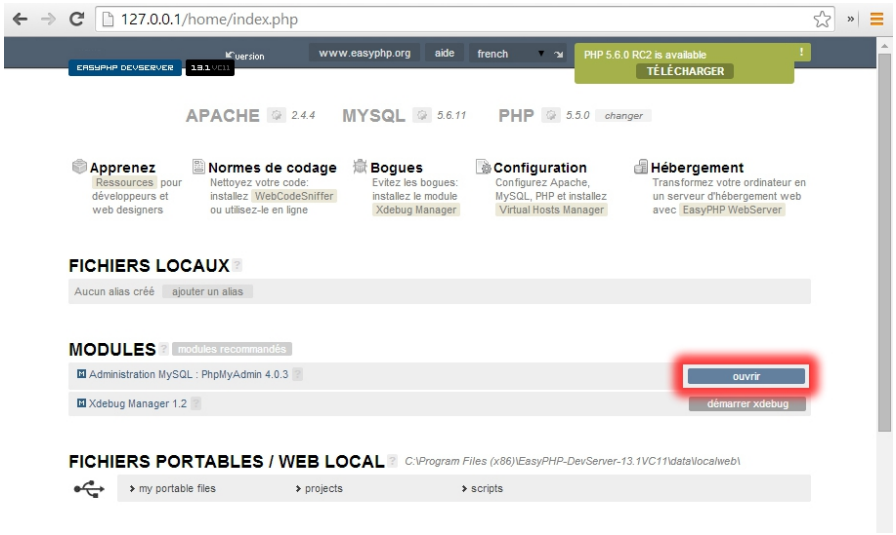
Attention : La mise en ligne du CMS WordPress implique une modification des liens inscrits dans la base de données. Des requêtes MySQL permettent d'effectuer ces changements. Ceci fait l'objet du chapitre « La migration du CMS » de ce document.

Étapes de l'installation de WordPress sur un serveur local (EasyPHP pour l'exemple)

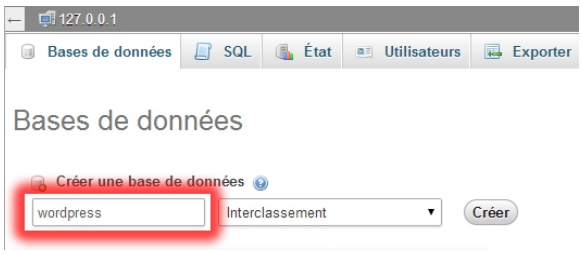
1. **Télécharger WordPress** depuis le site [http://wordpress.org] ou le site [http://fr.wordpress.org] pour la version francophone de l'administration du CMS.

2. **Décompresser le fichier wordpress.zip** dans le répertoire du serveur local (C:\Program Files\EasyPHP\data\localweb\projects\). Renommer le dossier au nom du projet (site).

3. **Démarrer le serveur local** (si ce n'est pas déjà fait). Accéder au serveur depuis un navigateur Web à l'adresse [http://127.0.0.1/home] ou [http://localhost/home]. Cliquer sur le bouton [ouvrir] du module Administration MySQL. Ceci permet d'accéder à l'outil d'administration de la base de données PhpMyAdmin.

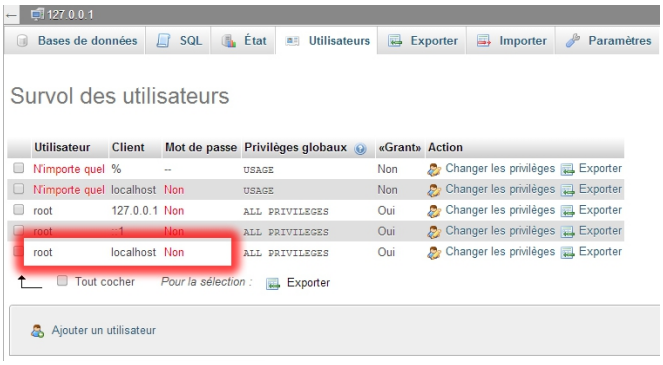


4. Dans l'outil d'administration PhpMyAdmin, cliquer sur l'onglet [Bases de données], puis **créer une base de données** en indiquant le nom de la base de données dans le champ réservé à cet effet (« wordpress », serait indiqué comme de base de données. Cliquer sur le bouton [Créer].



Il sera nécessaire de connecter le CMS à cette base. Il est par conséquent utile de créer un compte d'accès ou du moins observer ceux qui existent. Cliquer sur l'onglet « Utilisateurs » et choisir l'un des utilisateurs existants pour la connexion du CMS. Les informations suivantes seront utiles :

client (hôte) : **localhost**
nom d'utilisateur : **root**
mot de passe : ***(aucun)***



5. **Démarrage de l'installation du CMS.** Accéder à la page d'accueil du serveur [http://127.0.0.1/home] ou [http://localhost/home], puis cliquer sur le lien permettant d'accéder au CMS portant le nom du projet (nom indiqué précédemment lors du nommage du dossier décompressé du WordPress téléchargé).

WordPress informe que le fichier [wpconfig.php] n'est pas existant. Il sera créé lors de l'installation. Lancer l'installation en cliquant sur le bouton « Créer le fichier de configuration ». WordPress vous informera ensuite de relever les informations concernant la connexion à la base de données pour préparer les étapes qui vont suivre (déjà fait).

Indiquer ensuite **les informations concernant l'accès à la base de données**.

Ces informations ont été précédemment relevées depuis l'outil d'administration de la base de données PhpMyAdmin. Cliquer sur le bouton « Valider ».

<u>Note</u> : Le préfixe de table n'a pas a être modifié et peut éventuellement être supprimé.

Indiquer les informations relatives au futur site Internet. Il s'agit du nom du site et des accès à la zone d'administration du site. Il sera également nécessaire définir une adresse e-mail (celle-ci permettra d'envoyer un nouveau mot de passe en cas d'oubli...).

Cliquer sur le bouton « Installer WordPress ».

Notes : Ces informations pourront être modifiées depuis la zone d'administration une fois le CMS installé.

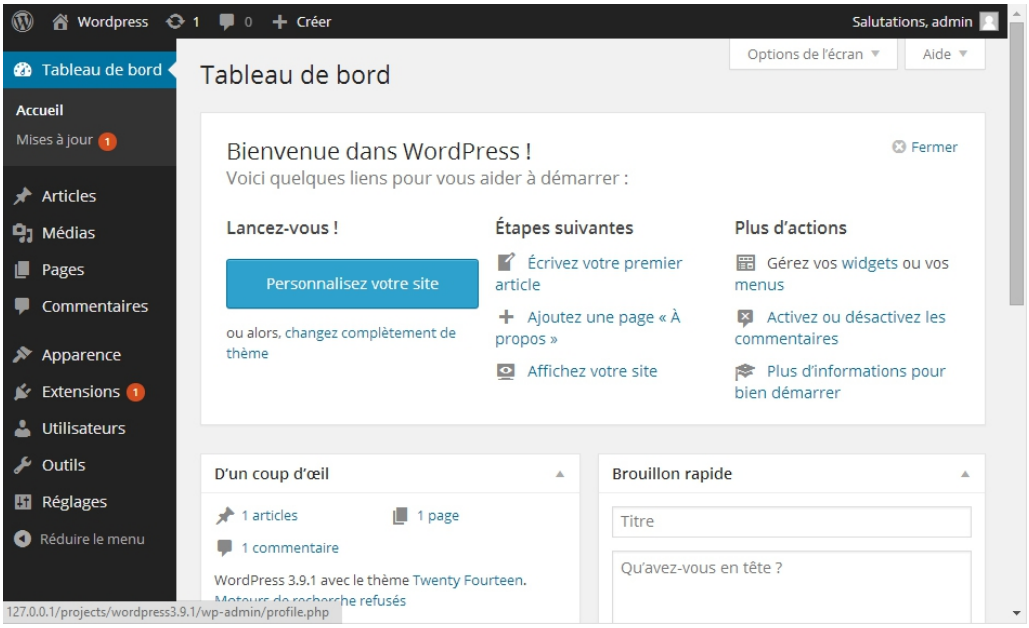
6. Le **CMS est installé** comme indiqué à la dernière étape du processus. Il est maintenant possible d'accéder à la zone d'administration en indiquant le nom d'utilisateur et mot de passe spécifiés lors de l'installation.

Il est dorénavant possible d'accéder au site en indiquant l'adresse [http://localhost/(répertoire du site)]. Il est également possible d'accéder directement à la zone d'administration depuis l'adresse [http://localhost/(répertoire du site)/wp-admin].

Premiers pas...

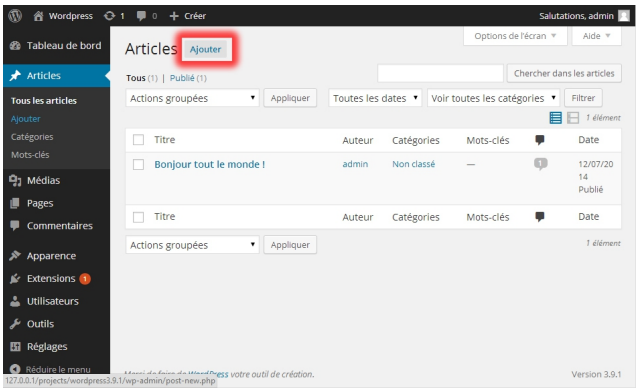
La zone d'administration dispose d'un menu latéral divisé en deux portions. La première partie permet de gérer les contenus (Articles, Médias, Pages et Commentaires) et la seconde les fonctionnalités et réglages du site (Apparence, Extensions, Utilisateurs, Outils et Réglages).

Rapide survol des fonctionnalités et outils qui apporteront de la modularité et de la flexibilité dans la création du thème et des contenus du site.

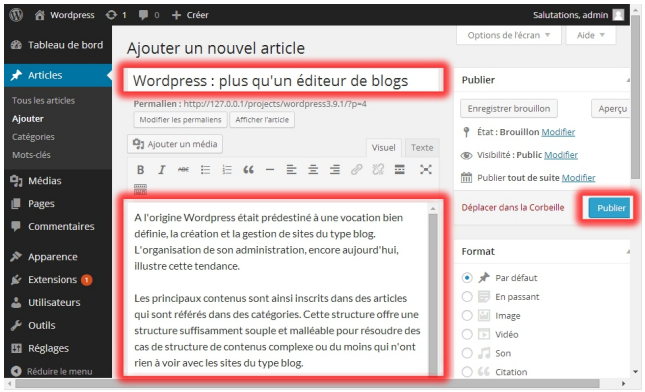


Les articles et catégories

Les articles sont la pierre angulaire du CMS. Ils permettent de créer des contenus, de les grouper grâce aux catégories.



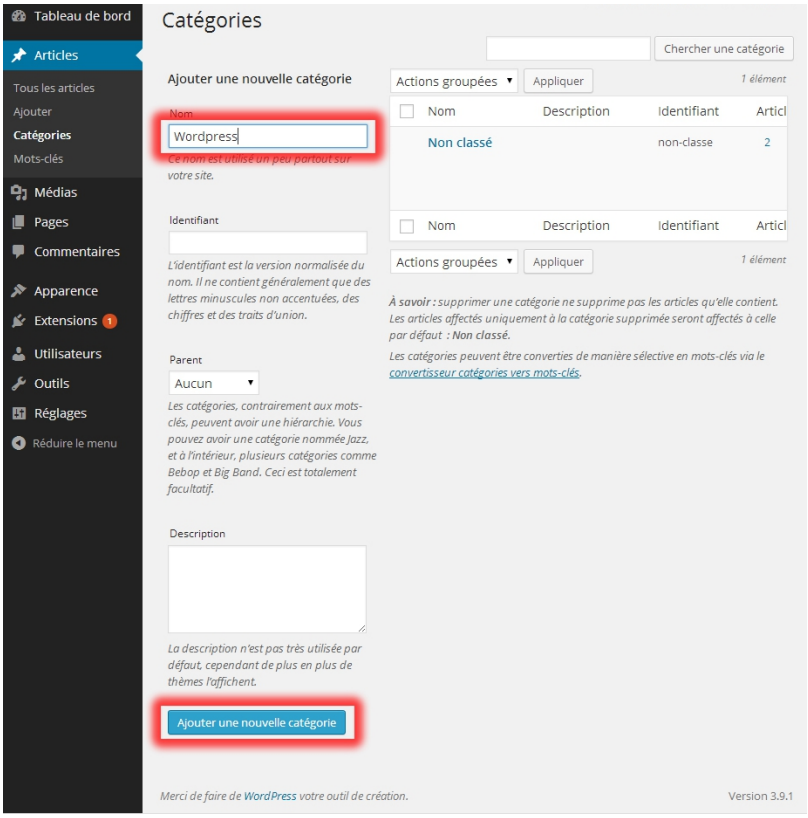
Ajouter un article



Création d'un article. Insertion d'un titre et du contenu. Affichage dans le site par le bouton « Publier ».

Les catégories permettent de regrouper les articles et de les classer. Le CMS réserve des fonctionnalités spécifiques qui s'avèrent très utiles et qui contribuent à la modularité du CMS :

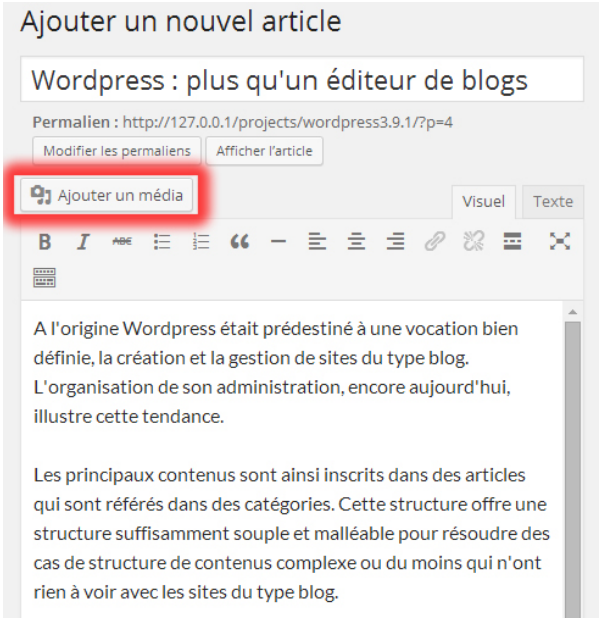
- Réserver l'affichage d'un groupe d'articles sur une page du thème qui leur est propre. Ceci se fait par des requêtes spécifiques dans le thème (tel qu'expliqué dans le chapitre « Les fonctions supplémentaires » de ce document.
- Ajouter des champs et fonctions spécifiques à une catégorie d'articles. Grâce notamment à l'extension « Advanced Custom Fields » comme expliqué dans le chapitre « Les extensions ».



Les catégories peuvent être hiérarchisées et ainsi hériter des fonctionnalités d'une catégorie parent.

Les médias

L'insertion d'images dans un article

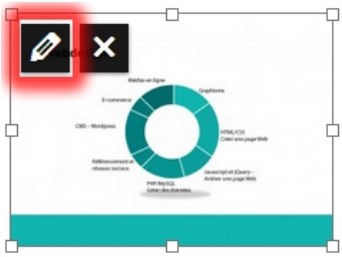


1. Ajouter une image

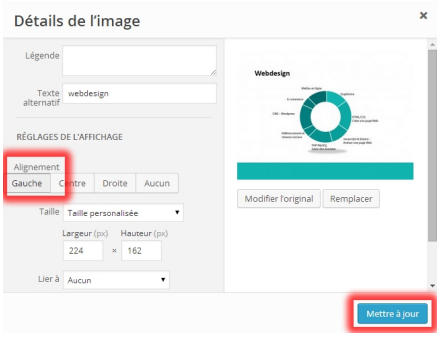
2. Sélectionner un fichier à importer dans la bibliothèque

3. Insérer l'image dans l'article

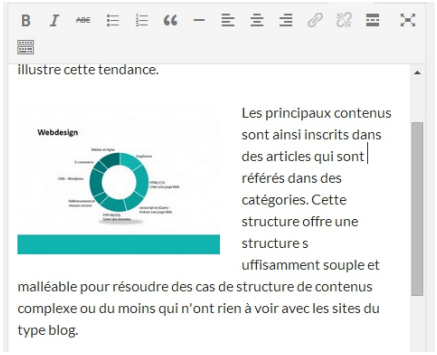
En survolant une image insérée dans un article, apparaît les boutons d'édition et de suppression de l'image. L'édition offre des options qui permettent d'aligner les images par rapport au texte ou de modifier la dimension de l'image.



1. Edition d'une image



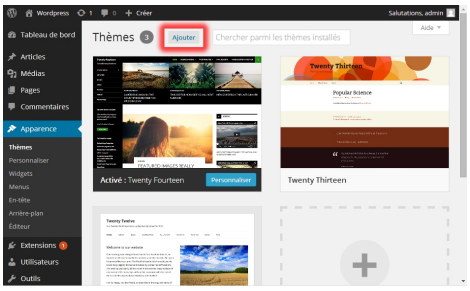
2. Modifier les paramètres de l'image



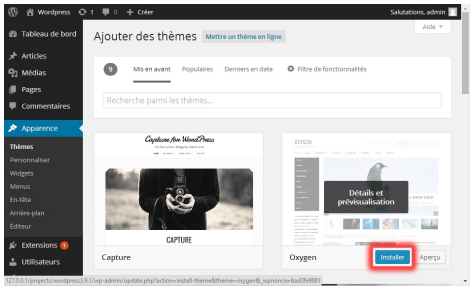
3. Résultat des modifications effectuées

Les thèmes

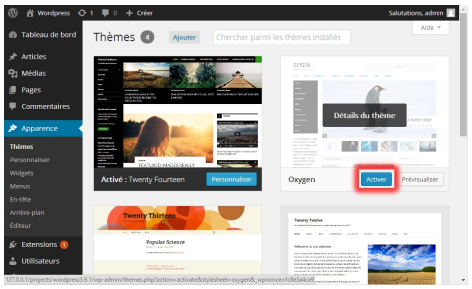
L'administration des thèmes consiste à définir l'apparence du site. Certaines fonctionnalités sont spécifiques au thème. Il est toutefois possible d'éditer les fichiers du thème afin d'y apporter de nouvelles fonctionnalités ou tout simplement imposer des changements. Attention toutefois, la modification d'un thème peut s'avérer complexe s'il est question d'apporter des changements qui vont au-delà des options visibles dans la zone d'administration. Il peut s'avérer plus simple de créer un thème spécifique de toute pièce...



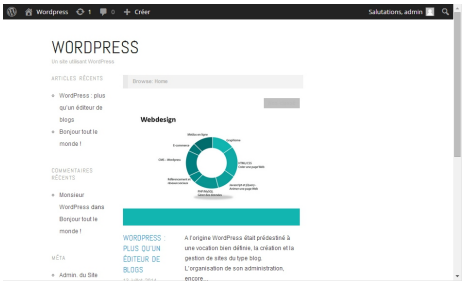
1. Ajouter un thème



2. Sélectionner le thème « Oxygen » qui sera téléchargé

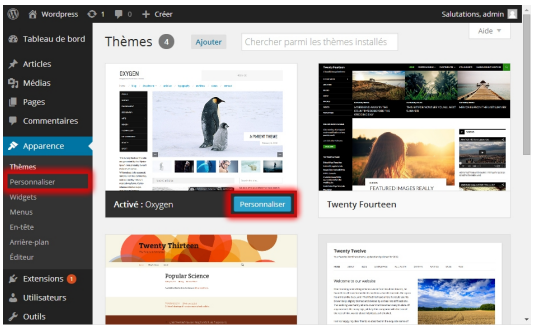


3. Activer le thème

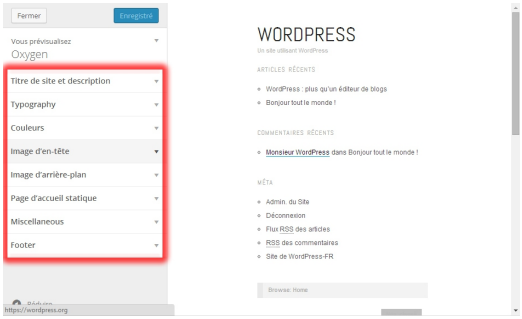


4. Résultat l'apparence du site adopte celle du thème

Les thèmes offrent des options qui leur sont spécifiques. L'exemple du thème choisi permet de modifier notamment l'apparence de la page d'accueil, des couleurs ou des polices de caractère du site.

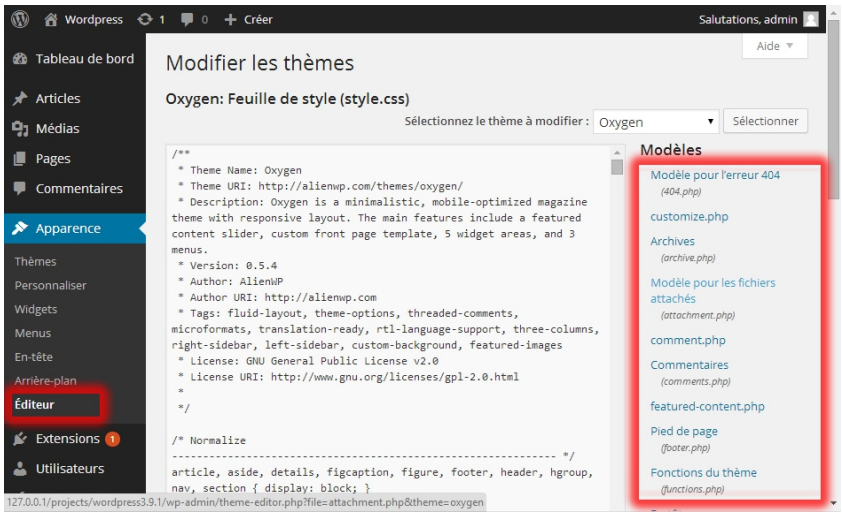


1. Cliquer sur « Personnalisation »



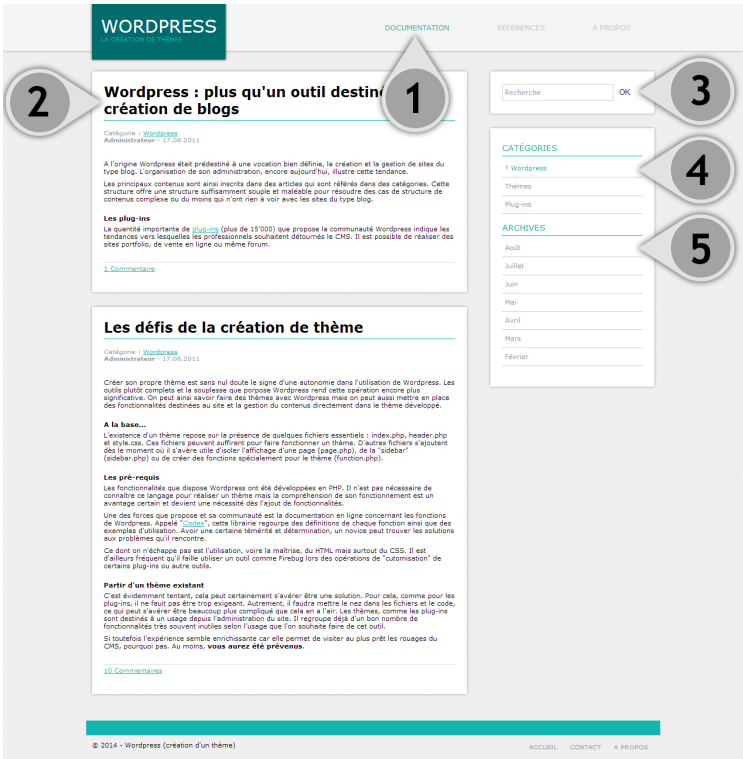
2. Une interface spécifique offre la possibilité de modifier les options du thème

Un éditeur de code est également disponible pour modifier les composants du thème. Cet éditeur accède à tous les fichiers qui composent le thème. Ce type d'opération peut cependant s'avérer laborieux. Il implique de comprendre les fonctions du thème et sa composition graphique (styles et pages).



La préparation à la création du thème

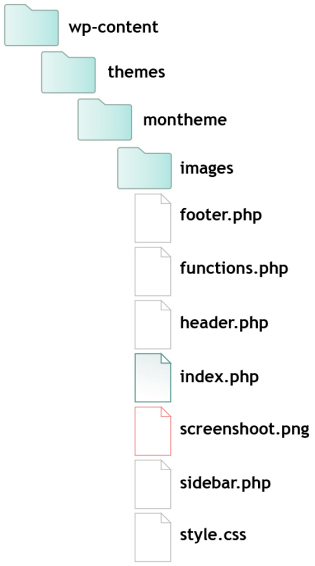
Des distinctions sont à faire au sujet des différents éléments qui composent l'interface tel que WordPress peut le comprendre. Voici une illustration sommaire.



1. **Pages** : WordPress permet de créer des pages statiques. A la différence des articles, elles ne s'insèrent pas dans des catégories. Elles sont, par défaut, directement utilisées dans les menus.
2. **Articles** : Les contenus insérés dans les articles s'archivent automatiquement sur le site selon leur date de publication (en ordre chronologique) ou en fonction des catégories (point 4) auxquelles ils appartiennent.
3. **Outil de recherche (widget)** : La recherche se fait dans les contenus du site.
4. **Catégories (widget)** : Classement des articles selon leur catégorie d'appartenance.
5. **Archives (widget)** : Classement des articles selon leur date de publication.

Les fichiers du thème

Les thèmes de WordPress sont répertoriés dans le répertoire [**wp-content>themes**].



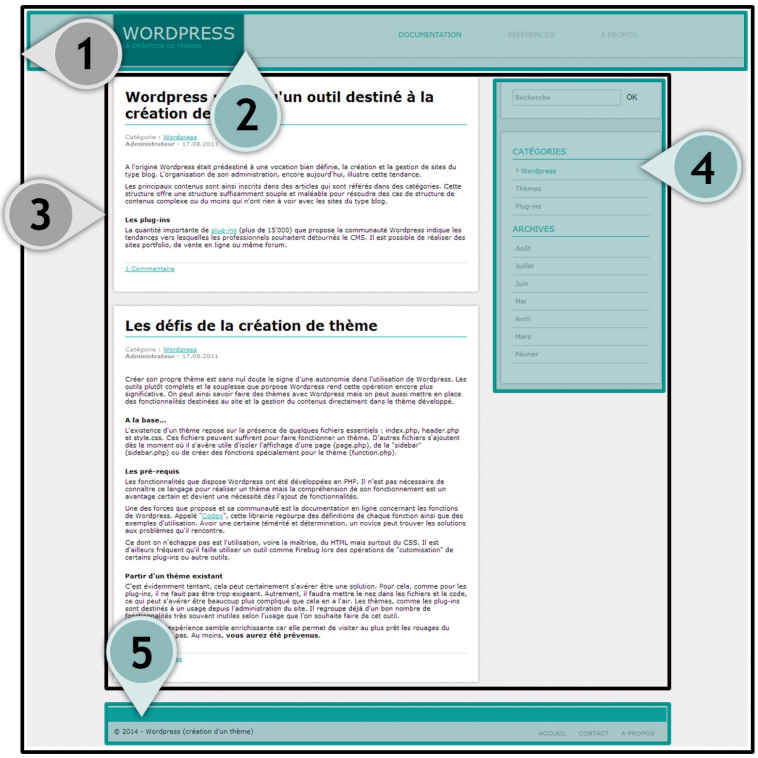
1. Créer dans [**wp-content>themes**], un répertoire qui contiendra le thème. Donner le nom du futur thème (ex. : 'montheme').
2. Créez dans ce répertoire les fichiers **index.php, header.php, footer.php, sidebar.php, functions.php** et **style.css** ainsi qu'un répertoire **images**.
3. Créer le fichier « screenshot.png ». Ce fichier est une capture d'écran en 880px X 660px de l'aspect graphique du thème.
4. Les informations suivantes sont à indiquer dans le fichier **style.css**

```
/*
Theme Name: Atelier WordPress
Theme URI: http://www.monsite.com/
Description: Description de ce theme.
Version: 0.1
Author: Prénom Nom
*/
```

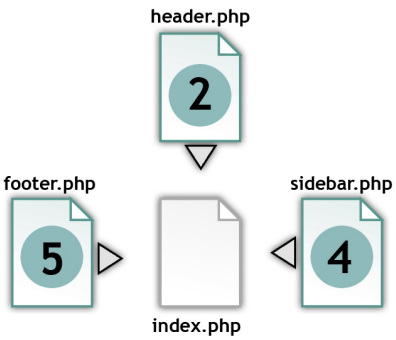
Note **:** En accédant à la rubrique 'Apparence' dans l'administration du site, on peut constater que le thème est déjà reconnu par le CMS. *Cette reconnaissance est possible grâce aux indications inscrites dans le fichier style*.css.

La composition d'un thème

La création d'un thème repose sur la manière dont les contenus sont utilisés et mis à jour. Ainsi, les éléments tels que l'entête, la barre de navigation secondaire (sidebar) ainsi que le pied de page sont séparés (isolés dans des fichiers qui leur sont propres) pour être utilisés par différents types de pages du thème ou sous-page (comme l'appelle WordPress) tels que présentés dans le chapitre « Augmenter les fonctionnalités du thème » de ce document. Le résultat est une association des fichiers qui permettent la composition de la page.



Les zones de la page



Les fichiers qui composent la page

1. **Zone globale** [index.php] : Elle regroupe l'ensemble de la structure et du contenu de la page. Elle fait appel aux autres zones qui s'insèrent dans des emplacements définis. Ultérieurement, ce modèle sera décliné pour les pages tels que page.php, single.php ou archive.php.
2. **Zone entête** [header.php] : Réunit les éléments graphiques qui concernent l'entête du site (souvent avec le menu principal) qui sera visible sur chaque page du site. Cette zone contient également les informations de la balise <head>.
3. **Zone contenu principal** [index.php] : Cette partie se trouve dans chaque sous-page du thème. Elle se définit dans le même fichier que la zone globale. Il s'agit du contenu principal en particulier pour les pages et les articles de Wordpress.
4. **Zone « sidebar »** [sidebar.php] : Cette zone regroupe les outils complémentaires du site que propose WordPress. A priori, WordPress fait référence à cette zone comme étant un des emplacements clés pour insérer des « widgets ».
5. **Zone pied de page** [footer.php] : Zone du pied de page commune à toutes les pages du site.

Les étapes de la création d'un thème

L'opération de la création du thème implique plusieurs étapes. L'ordre dans l'exécution de ces étapes peut être déterminant et assure la cohérence de l'opération. On considère dans les étapes qui suivent qu'**une maquette en HTML et CSS existe** déjà et qu'elle dispose sans doute des styles pour les supports mobiles.

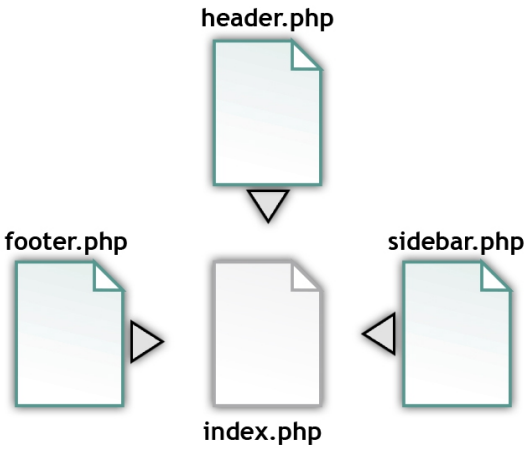
Certaines étapes ne sont pas essentielles au fonctionnement du thème. Elles dépendent de l'ampleur des fonctionnalités et de la modularité prévue pour le thème.

1. **Passer les images de la maquette au thème**
   Cette première étape consiste à simplement transférer les images qui composent la maquette du site dans le répertoire « images » du thème. Si la maquette dispose de polices de caractères spécifiques (Web fonts), la même opération est à prévoir. Un dossier « fonts » sera à créer.

2. **Le transfert des styles CSS**
   Cette étape consiste à simplement copier-coller les styles de la feuille de style de la maquette vers le fichier « style.css » du thème. Ceci conditionnera l'apparence du thème qui prendra de plus en plus forme après chaque étape.

   Ne pas oublier de modifier les liens vers les fichiers images pour les propriétés `background-image` et `list-style-image`.

3. **Composer la mise en page HTML**
   Cet aspect implique de bien comprendre la composition d'un thème (introduit au précédent chapitre). Ainsi, le résultat de l'affichage du thème est le résultat de l'inclusion au fichier « index.php » des fichiers « header.php » et accessoirement « sidebar.php » et « footer.php ».



   L'étape-ci implique de poser les balises HTML provenant de la maquette. Dans un premier temps celles de la composition de la page HTML `<?doctype>`, `<html>`, `<header>` (et son contenu) et `<body>`.
   Ensuite introduire les balises utiles à la mise en page. Généralement composée de `<div>`, `<header>`, `<main>`, `<section>`, `<aside>` et `<footer>`. Ne pas oublier les `id` et `class` qui donnent la mise en forme CSS de cette structure. Cette étape est détaillée dans le chapitre « codes PHP de Wordpress » de ce document.

4. Introduire les fonctions PHP de WordPress

Il n'est pas question de faire du développement en PHP mais de comprendre ce que les fonctions PHP de WordPress permettent d'introduire dans le thème. L'appel d'une fonction comme `<?php bloginfo('name'); ?>` introduit le nom du site dans le thème, à l'endroit où la fonction a été posée. Il est par conséquent important d'appliquer ces fonctions dans les balises HTML prévues pour leur donner la forme graphique prévue par le CSS.

Les principales fonctions PHP de WordPress sont présentées dans le chapitre suivant.

HTML

```
<h1>
<a href="<?php bloginfo('url'); ?>">
        <?php bloginfo('name'); ?><br />
        <span><?php bloginfo('description'); ?></span>
</a>
</h1>
```

CSS

```
h1 {
        font-size:30px;
        height:50px;
        font-weight:normal;
        background-color:#006c6c;
        text-transform:uppercase;
        padding:30px 15px 15px 15px;
        line-height:15px;
        box-shadow:0px 0px 5px #333;
}
h1 a, h1 a:hover{
        text-decoration:none;
        color:#fff;
}
h1 a span{
        font-size:10px;
        color:#12b5b0;
        text-transform:uppercase;
}
```

RÉSULTAT DANS LE NAVIGATEUR

5. L'introduction de la boucle

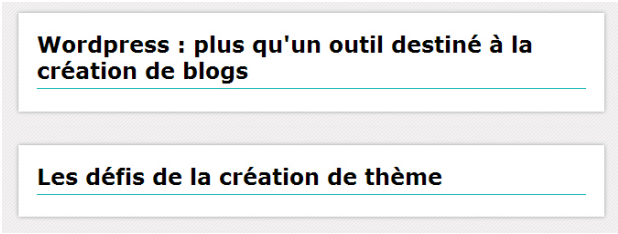
Il s'agit de l'élément clé du processus qui permettra d'introduire le contenu **des articles et des pages**. La notion de boucle peut sembler abstraite. Elle évoque la répétition d'un affichage autant de fois que des articles auront été publiés.

Voici un exemple simplifié de la boucle de WordPress introduite dans une page.

HTML

```
<?php while(have_posts()) : the_post(); ?>
   <article>
      <h2><?php the_title(); ?></h2>
   </article>
<?php endwhile; ?>
```

RÉSULTAT DANS LE NAVIGATEUR

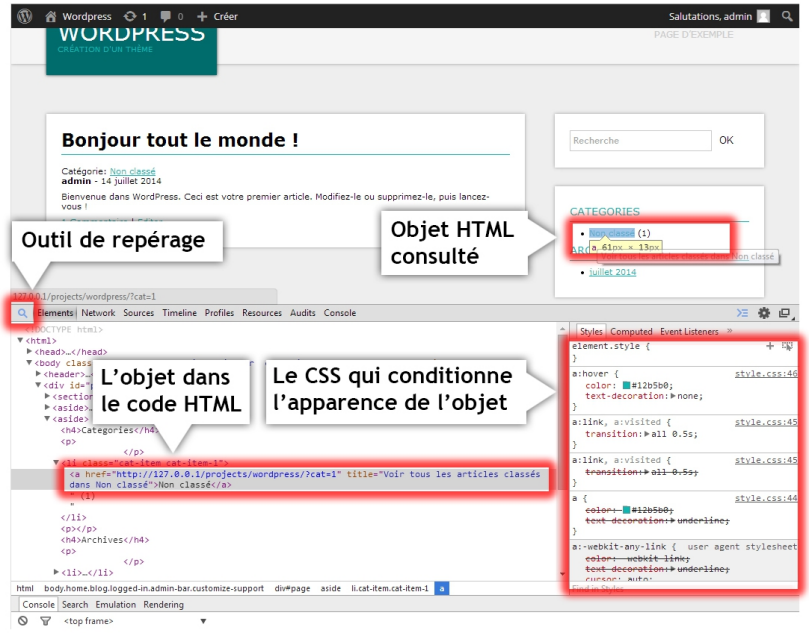


Dans cet exemple, la boucle génère une balise `<article>` disposant d'un titre dans une balise `<h2>` pour chaque article publié. La fonction `<?php the_title(); ?>` introduit le titre de chaque article. Ce processus automatise l'affichage des contenus. Toutes les informations d'un article ou d'une page peuvent être introduites dans ces boucles. Elles font appel à des fonctions PHP de WordPress spécifiques qui seront expliqués dans le prochain chapitre.

6. **Corrections CSS**

WordPress n'applique pas nécessairement les CSS prévus de la maquette préalablement conçue. D'ailleurs, WordPress génère des objets HTML avec des classes CSS qui lui sont propres que ce soit pour l'affichage de menus, de widgets ou de la mise en fonction des extensions. Par conséquent, à partir de cette étape, les corrections CSS s'appliquent jusqu'au terme de la création du thème.

L'outil à employer dans cette opération est l'outil de développement, introduit dans les navigateur (F12 sur PC) ou Firebug pour Firefox. Ces outils permettent de repérer rapidement les objets HTML qui composent l'interface et les CSS qui conditionnent leur apparence.



Outil de développement du navigateur Google Chrome

7. **Mise en fonction des menus**

L'appel de la fonction PHP de WordPress `<?php wp_nav_menu(); ?>` permet d'introduire un menu qui est automatiquement pris en charge par le CMS en affichant les pages publiées et créées depuis l'administration du CMS. Cette fonction peut cependant être personnalisée et faire appel à des menus composés sur mesure depuis l'administration du CMS. L'appel de cette fonction sera dans ce cas fait à chaque emplacement dans le thème où un menu devra apparaître.

Une explication détaillée est présente dans le chapitre concernant « La gestion des menus ».

8. **Mise en fonction des Widgets**

De la même façon que pour les menus, il est possible d'introduire une ou plusieurs zones d'affichage de Widgets dans le thème.

Une explication détaillée est présente dans le chapitre concernant « La gestion des Widgets ».

9. **Création des sous-pages**

   Le fichier « index.php » est utilisé pour afficher toutes les pages du site. Il peut s'avérer difficile de supporter la mise en page et affichage des contenus pour toutes les pages du site à travers un seul gabarit, en l'occurrence « index.php ». Il suffit de penser à la page d'accueil d'un site. Celle-ci est très souvent différente des autres pages. Raison pour laquelle il existe les sous-pages (« home.php », dans ce cas).

   Ces sous-pages font partie du thème. WordPress a prévu une série de fichiers définis qui seront utilisés à la place de « index.php » si elles ont été créées dans le thème. Il existe notamment la possibilité de créer un fichier « home.php » qui, s'il est présent, sera employé pour l'affichage de la page d'accueil. Ce fichier pourra également faire appel au fichier « header.php », « sidebar.php » ou « footer.php », comme le fait « index,php ».

   Une explication détaillée est présente dans le chapitre « Augmenter les fonctionnalités du thème ».

10. **Création des boucles spécifiques**

    Par défaut, la boucle que propose WordPress affiche les contenus des pages et des articles. Il peut s'avérer utile, voire nécessaire de recueillir des contenus spécifiques (les articles d'une catégorie, par exemple) et de les afficher à un endroit de la page. Cette opération implique de créer une requête spécifique avec la fonction PHP de WordPress `<?php query_posts( 'cat=3' ); ?>` qui fera une sélection des contenus avant de les afficher dans la boucle.

    Une explication détaillée est présente dans le chapitre « Les fonctions supplémentaires utiles » concernant l'utilisation des requêtes de WordPress.

11. **Introduction de nouvelles fonctions grâce aux extensions**

    Les extensions permettent d'introduire des fonctionnalités supplémentaires au CMS qui peuvent également impliquer l'insertion de code dans le thème. L'extension « Advanced Custom Fields » permet d'ajouter des champs aux articles. Afin d'introduire les contenus de ces champs, des codes spécifiques doivent être introduits dans le thème.

    Une introduction est présentée dans le chapitre « Les extensions ».

12. **Développement de fonctions supplémentaires (avancé)**

    Le fichier « functions.php » permet de communiquer avec l'administration du CMS et d'introduire des fonctionnalités propres au thème créé. Ceci implique cependant de disposer de connaissances dans le développement PHP, mais permet d'introduire des fonctionnalités supplémentaires et faire en sorte que des options concernant la mise en forme du thème puissent être modifiées et personnalisées depuis la zone d'administration du CMS.

header.php

Y figure les informations inscrites dans la balise <head> et la partie du code HTML qui se répétera systématiquement sur toutes les pages du site. Constat important à faire à ce stade, la fermeture de la balise </body> n'est pas présente dans le document.

```
<!doctype html>
<html>
<head>
   <meta charset="<?php bloginfo('charset'); ?>" />
   <title><?php wp_title('|'); ?> <?php bloginfo('name'); ?></title>
   <link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>" type="text/css" />
   <link rel="pingback" href="<?php bloginfo('pingback_url'); ?>" />
   <?php wp_head(); ?>
</head>
<body <?php body_class(); ?>>
   <a href="<?php bloginfo('url'); ?>"><?php bloginfo('name'); ?></a>
   <?php bloginfo('description'); ?>
   <?php wp_nav_menu(); ?>
```

Les fonctions PHP utilisées dans ce fichier

`bloginfo()` Cette fonction dispose des informations générales du site. On peut, par exemple, obtenir le nom du site, l'adresse absolue du site ou la description du site.

- `bloginfo('charset')` : Indique le format d'encodage des caractères.
- `bloginfo('stylesheet_url')` : l'adresse absolue de la feuille de style « style.css » du thème.
- `bloginfo('pingback_url')` : Le pingback permet de signaler qu'un site externe se connecte à un article (en créant un lien vers la page d'un article du site). WordPress sait détecter ce tye de lien (plus d'infos : http://www.woodymood-dev.net/cms/wordpress/fr/2009/01/26/pings/).
- `bloginfo('url')` : indique l'adresse absolue du site.
- `bloginfo('name')` : indique le nom du site.
- `bloginfo('description')` : indique la description du site.
- `bloginfo('template_url')` : l'adresse absolue du thème.

`wp_title()` Affiche le titre de la page courante.

- `wp_title('|')` : ajoute le séparateur indiqué.

`wp_head()` Positionnée juste avant la fermeture de la balise </head>. Ajoute les liens aux fichiers CSS et Javascript necessaires notamment au fonctionnement des plugins.

`body_class()` Intègre les classes (CSS) de Worpress et des futurs extensions du site.

`wp_nav_menu()` Affiche un ou des menus WordPress. **Par défaut**, cette fonction se réfère à la liste des pages créées depuis la zone d'administration. Il est possible de créer plusieurs menus. Pour cela, voir le chapitre "La gestion des menus" de ce document.

Index.php

Le fichier index.php a deux rôles :

1. Celui d'assembler les zones (header.php, sidebar.php, footer.php).

```
<?php get_header(); ?>

<?php get_sidebar(); ?>

<?php get_footer(); ?>
```

2. Celui d'afficher les articles (dans une boucle).

   La boucle qu'utilise WordPress permet d'afficher les articles et les pages.

```
<?php get_header(); ?>

<?php if(have_posts()) : ?>

   <?php while(have_posts()) : the_post(); ?>
       <article <?php post_class(); ?>>
           <h2>
               <a href="<?php the_permalink(); ?>" title="<?php the_title(); ?>">
                   <?php the_title(); ?>
               </a>
           </h2>

          <p class="article_entete">
           Cat&eacute;gorie: <?php the_category(', ') ?><br />
               <strong><?php the_author() ?></strong> - <?php the_time('j F Y') ?>
          </p>

           <?php the_content(); ?>

           <p class="commentaire">
               <?php comments_popup_link('Aucun', '1 Commentaire', '% Commentaires'); ?>
               <?php edit_post_link('Editer', ' &#124; ', ''); ?>
           </p>
       </article>
   <?php endwhile; ?>

    <div>
        <?php wp_link_pages( array(
                'separator' => ' - ',
                'nextpagelink' => 'suivant',
                'previouspagelink' => 'pr&eacute;c&eacute;dent'
                )
            ); ?>
    </div>

<?php else : ?>

   <article>
       <h2>Page introuvable</h2>
       <p>Cette page n'existe pas. Nous regrettons du désagrément.<p>
   </article>

<?php endif; ?>

<?php get_sidebar(); ?>

<?php get_footer(); ?>

<?php wp_footer(); ?>

</body>

</html>
```

Les fonctions utilisées dans ce fichier

```
if(have_posts()) :
else :
endif;
```

Cette condition vérifie si un ou des articles existe(nt). Dans le cas contraire, il est possible de préciser qu'aucun contenu n'a été trouvé.

```
while(have_posts()) : the_post();
endwhile;
```

La boucle qui affichera chaque article trouvé. Les fonctions qui suivent doivent être insérées dans cette boucle pour générer du contenu.

```
the_permalink();
```

Affiche l'adresse absolue vers l'article.

```
post_class();
```

Affiche les classes CSS de Wordpress définies pour chaque article.

```
the_title();
```

Affiche le titre de l'article.

```
the_id();
```

Affiche l'id de l'article tel que référencé dans la base de données. Ceci peut s'avérer pratique lors de la création de liens vers un élément de cet article.

```
the_category(', ')
```

Affiche les catégories dans lesquelles l'article est inséré. Chaque nom de catégorie est un lien vers la page qui affiche les articles appartenant à cette catégorie.

```
the_author()
```

Affiche le nom de l'auteur.

```
the_time('j F Y')
```

Affiche la date et l'heure de publication de l'article. Un ajustement du format d'affichage se définit dans les paramètres de la fonction.

```
comments_popup_link(
'Aucun',
'1 Commentaire',
'% Commentaires');
```

Affiche un lien vers les commentaires de l'article ainsi que le nombre de commentaires dont dispose l'article. Il est possible de définir la syntaxe pour chaque nombre de commentaires.

```
edit_post_link(
'Editer',
' &#124; ',
'');
```

Affiche un lien vers la zone d'administration du site pour modifier l'article. N'est visible que lorsque l'utilisateur est connecté.

```
wp_link_pages(
        array(
'separator' => ' - ',
'nextpagelink' => 'suivant',
'previouspagelink' =>
'pr&eacute;c&eacute;dent'
        )
);
```

Affiche la navigation entre les pages lorsque le nombre d'articles excède ce qui est paramétré das l'administration du CMS.

```
wp_footer();
```

Positionné juste avant la fermeture de la balise </body>. Ajoute les liens aux fichiers CSS et Javascript necessaires notamment au fonctionnement d'extensions.

sidebar.php

Le fichier sidebar.php est prédestiné à recevoir les Widgets qui figureront dans le thème. Il existe deux méthodes pour afficher les Widgets.

Fixer les Widgets dans le thème

En les définissant de manière permanente en utilisant le code PHP de WordPress qui fait directement appel aux Widgets désirés.

Le formulaire de recherche

```
<form method="get" id="searchform" action="<?php bloginfo('home'); ?>/">
<input type="text" value="<?php the_search_query(); ?>" name="s" id="s" />
     <button>OK</button>
</form>
```

Le calendrier

```
<?php get_calendar(); ?>
```

Les catégories

```
<?php wp_list_categories(array('order'=>'ASC', 'hierarchical'=>true)); ?>
```

- **'order'=>'ASC=** permet un tri par ordre alphabétique,
- **'hierarchical'=>true** indique l'affichage des sous-catégories. **false** indique de ne pas afficher les sous-catégories.

Consulter la documentation en ligne pour connaître les autres options qu'offre cette fonction.

Les pages statiques

```
<?php wp_list_pages('title_li'=>'<h2>Pages</h2>'); ?>
```

Indique que le titre de la liste statique doit figurer dans un <h2>.

Les archives

```
<?php wp_get_archives('type=monthly'); ?>
```

Le type=monthly indique que les archives sont classées par mois.

La liste des liens utiles

```
<?php get_links_list(); ?>
```

Liste des sites de référence géré dans la zone d'administration [Liens].

Les flux RSS

```
<a href="<?php bloginfo('rss2_url'); ?>">Flux RSS - Articles</a>
<a href="<?php bloginfo('comments_rss2_url'); ?>">Flux RSS - Commentaires</a>
```

Suivre à distance les nouveaux articles et les récents commentaires ajoutés dans le site.

Enregistrement

```
<?php wp_register(); ?>
```

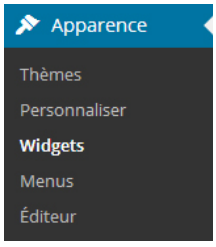
Lien vers la zone d'administration du site

Déconnexion

```
<?php wp_loginout(); ?>
```

Permet de se déconnecter et fermer une session ouverte.

Dynamiser l'administration des Widgets



WordPress permet également une gestion dynamique des Widgets depuis la zone d'administration sans avoir à intervenir directement dans les fichiers du thème. Il sera ainsi possible de gérer depuis la zone d'administration les Widgets ainsi que leur positionnement et leur configuration.

1. Appel à la fonction PHP de l'administration des Widgets

functions.php

Un fichier « **functions.php** » doit être créé (s'il n'existe pas déjà) dans le thème. Ajouter la fonction `register_sidebar()` qui établit la liaison entre le thème et la gestion des Widgets dans la zone d'administration du CMS. Dans l'état, la fonction n'opère que sur une seule colonne. La disposition de Widgets sur plusieurs colonnes est abordé dans le chapitre concernant « La gestion des Widgets ».

```php
<?php
if ( function_exists('register_sidebar') ){
        register_sidebar();
}
?>
```

2. Indiquer l'emplacement dynamique des Widgets dans le thème

sidebar.php

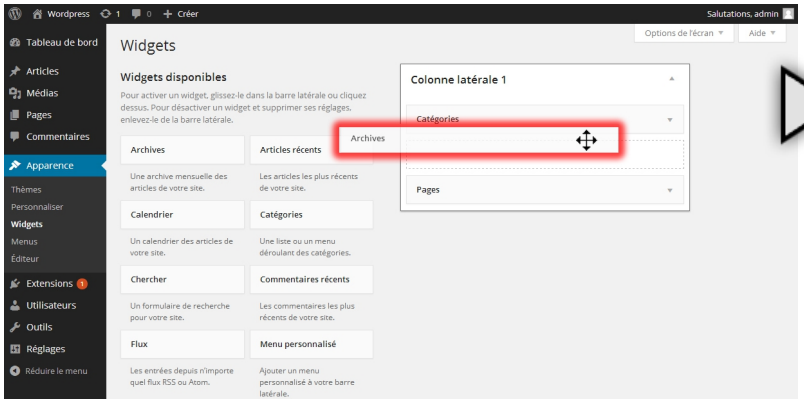
Il sera finalement nécessaire d'intervenir dans le fichier « **sidebar.php** », en vérifiant si la fonction `dynamic_sidebar()` existe. Cette fonction est créée par WordPress lorsqu'un administrateur active des Widgets. En la positionnant dans une condition, comme dans l'exemple ci-dessous, elle permet surtout d'introduire du contenu dans le cas où aucun Widget n'est activé depuis l'administration. On peut également indiquer des Widgets présents par défaut. Ce qui évite d'avoir une zone vide de contenu.

```php
<?php if ( !function_exists('dynamic_sidebar') || !dynamic_sidebar() ) : ?>
        // Le code PHP de WordPress faisant appel aux Widgets: Ils apparaîtront
        // dans le cas où aucun Widget n'est défini dans la zone d'administration
<?php endif; ?>
```

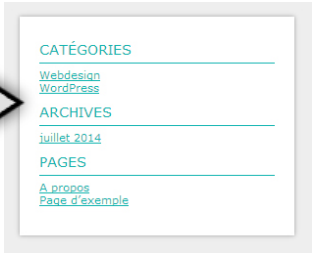
3. Gérer les Widgets depuis l'administration du CMS

L'étape-ci consiste à activer depuis la zone d'administration du CMS les Widgets qui seront présents dans la zone défini dans le thème.

ADMINISTRATION DU CMS



RÉSULTAT DANS LE THÈME



La gestion du thème depuis l'administration du CMS

Créer un thème qui permet d'afficher les contenus des articles et pages édités depuis l'administration du CMS est un objectif qui peut suffire à un site du moment où les intentions à l'égard du thème sont précises et limitées. Wordpress offre également la possibilité d'aller plus loin en apportant des modifications au thème depuis la zone d'administration du CMS et ainsi de le rendre encore plus maléable par les administrateurs. Cela peut correspondre par exemple à définir la composition de menus ou de zones disposant d'extensions (widgets) ou encore à la modification des couleurs de l'interface ou de l'image de l'entête. Une liaison est ainsi nécessaire entre le thème et l'administration. Comme précisé dans ce qui suit, le fichier **functions.php** tient justement ce rôle.

functions.php

Le fichier « **functions.php** » du thème permet d'introduire des fonctionnalités propres au thème et de les rendre disponibles dans l'administration du CMS. Des fonctions PHP sont mises à disposition par Wordpress à cette fin. Elles surclasseront les fonctionnalités définies par défaut dans le CMS pour laisser place aux paramétrages sur mesure.

Remplacer le comportement d'une fonction PHP du CMS

Pour modifier le comportement d'une fonction propre au CMS il suffit d'utiliser la fonction `add_filter()`. Celle-ci appelle la fonction à modifier et lui indique les transformations à opérer.

```php
<?php
add_filter( 'the_title', function( $title ){ return '<b>' . $title . '</b>'; } );
?>
```

*La fonction **add_filter()** : https://codex.wordpress.org/Function_Reference/add_filter*

Créer un comportement spécifique

Pour ajouter un comportement pendant l'exécution d'une fonction il suffit d'utiliser la fonction `add_action()`.

```php
<?php
add_action( 'wp_head', function(){ echo '<meta name="author" content="me" />'; } );
?>
```

*La fonction **add_action()** : https://codex.wordpress.org/Function_Reference/add_action*
La liste des actions disponibles : https://codex.wordpress.org/Plugin_API/Action_Reference

Ajouter des paramétrages prédéfinis au thème

Wordpress dispose de paramétrages prêt à l'emploi destinés aux thèmes qui peuvent être réglés depuis la zone d'administration. Ceux-ci s'activent à l'aide de la fonction `add_theme_support()`.

```php
<?php
add_theme_support('custom-background');
?>
```

*La fonction **add_theme_support()** : https://codex.wordpress.org/Function_Reference/add_theme_support*

Selon le paramètre indiqué dans la fonction, des réglages supplémentaires seront proposés dans l'administration du CMS. Il sera ainsi possible d'ajouter dans le thème des codes qui permettront que les changements réalisés par les administrateurs soient effectués.

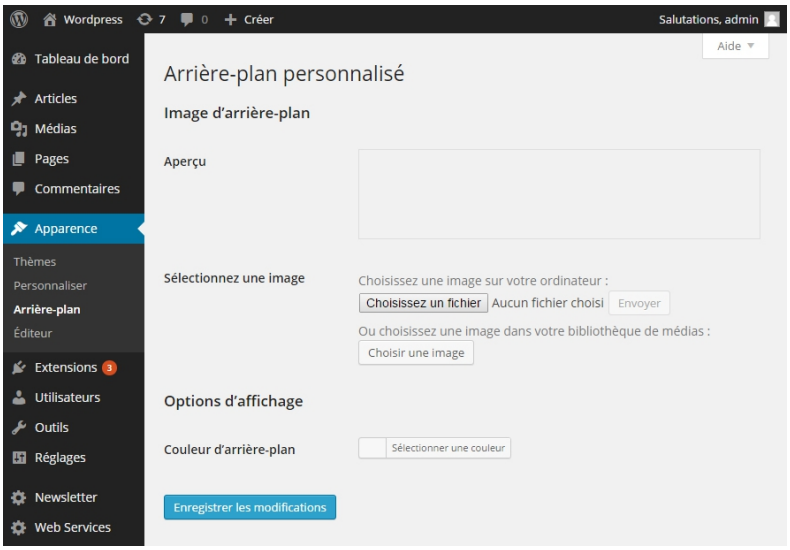


Image à la une

L'image à la une est une fonctionnalité qui permet d'ajouter depuis les articles édités dans la zone d'administration une image qui accompagnera l'article selon l'emplacement qui lui est réservé dans le thème.

1. **Appel à la fonction PHP de l'administration des menus**

   Cette fonctionnalité s'intègre dans le fichier « **functions.php** » du thème comme présenté précédemment, à l'aide de la fonction `add_theme_support()` en lui spécifiant comme paramètre `'post-thumbnails'`.

```php
<?php
add_theme_support('post-thumbnails');
?>
```

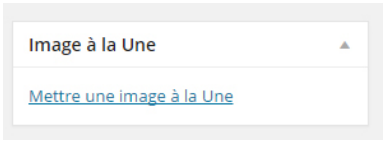
2. **Indiquer l'emplacement des images dans le thème**

   Comme indiqué chaque article ou page pourra disposer d'une image de référence. Pour la faire apparaître, le code suivant doit figurer dans la boucle PHP exploitant les contenus des articles et pages publiés.

```php
<?php
if( has_post_thumbnail() ) {
        the_post_thumbnail();
}
?>
```

3. **Gérer les images à la une depuis l'administration**

   Les images peuvent être ajoutées depuis l'administration du CMS à l'aide de la rubrique Image à la Une qui apparaît dans l'édition des articles et des pages.



La gestion des menus

On emploie la fonction `wp_nav_menu()` pour l'insertion d'un menu dans le thème. Cette fonction ne prend cependant en considération que les pages actives dans l'administration du CMS et compose automatiquement le menu. Il peut s'avérer nécessaire de disposer de plusieurs menus et que leur contenu soit composé sur mesure. La même fonction `wp_nav_menu()` dispose de cette capacité et permet ainsi la composition de menus disposant de pages, catégories ou liens gérés depuis la zone d'administration du CMS.

1. **Appel à la fonction PHP de l'administration des menus**

   Dans le fichier « **functions.php** » sont à citer les positions des menus du thème dans la fonction `register_nav_menus()` qui rend possible l'administration des menus par le CMS. Une dénomination devra être définie pour chaque menu existant. Celle-ci sera utilisée par l'administration du CMS pour identifier les menus du thème.


```php
<?php
register_nav_menus(
   array(
          'menu-principal' => __( 'Main menu' ),
          'menu-pied' => __( 'Footer menu' )
   )
);
?>
```

Nom du menu dans l'administration

Nom de la position

2. **Indiquer l'emplacement dynamique des menus dans le thème**

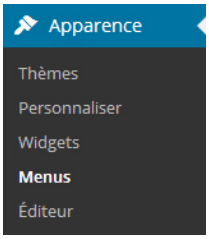
   Les menus pourront être indiviuellement appelés dans le thème en utilisant la fonction `wp_nav_menu()` aux emplacements souhaités. Des paramètres doivent être introduits lors de l'appel de la fonction comme le **nom de la position** qui est définit par `'theme_location'`. Le nom indiqué permettra d'établir la liaison avec l'administration du CMS à l'étape suivante. Cette fonction propose de nombreux paramètres. Pour en savoir plus une visite à la page (http://codex.wordpress.org/Function_Reference/wp_nav_menu) s'impose.

```php
<?php
// Appel d'un premier menu (le menu principal du site)
wp_nav_menu( array(
          'theme_location' => 'menu-principal'
) );
?>
```

Nom de la position

```php
<?php
// Appel d'un second menu (un menu en pied de page)
wp_nav_menu( array(
          'theme_location' => 'menu-pied'
) );
?>
```

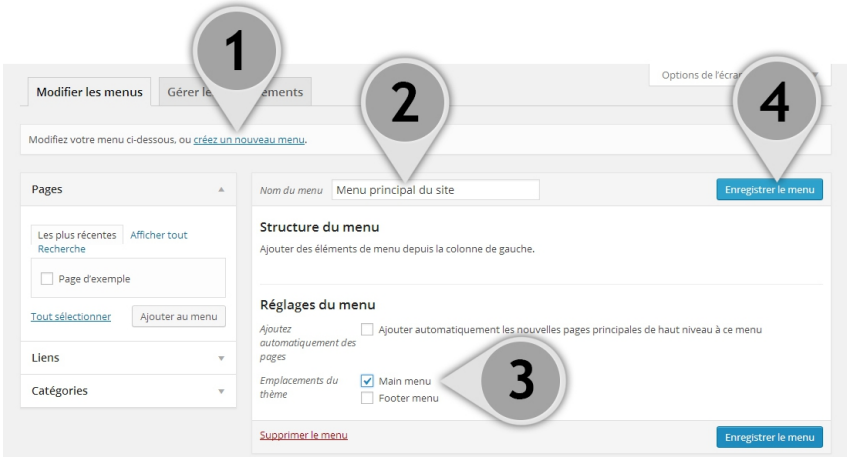
3. Gérer les menus depuis l'administration



Les menus cités dans la fonction `register_nav_menus()` peuvent dorénavant être assignés dans l'administration du CMS.

Ajouter un menu

L'opération consiste à assigner leur position aux menus du thème.

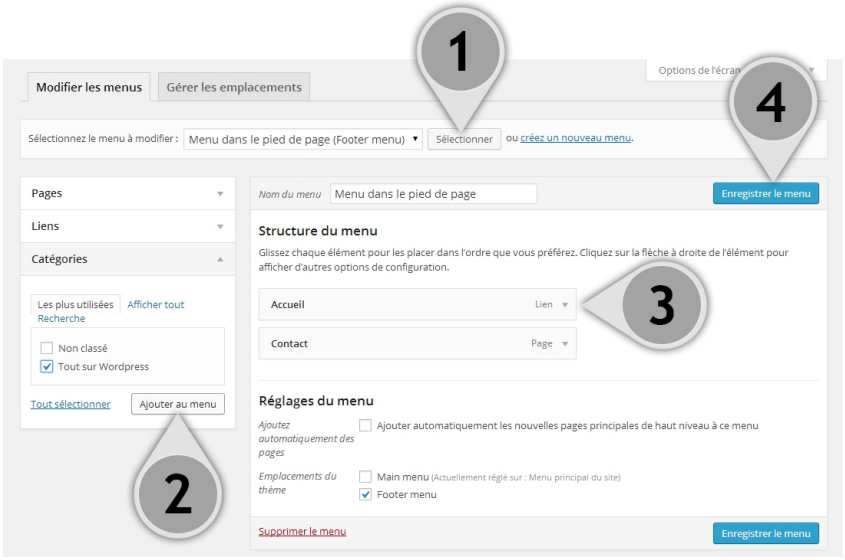


Gestion des menus dans la zone d'administration

1. Pour ajouter un menu, il faut cliquer sur le lien « créer un nouveau menu ».
2. Indiquer le nom du menu.
3. Assigner la position au menu apparaissant dans le thème. Les noms indiqués dans la liste sont ceux définis préalablement dans le fichier « functions.php ».
4. Cliquer sur le bouton « Enregistrer le menu ».

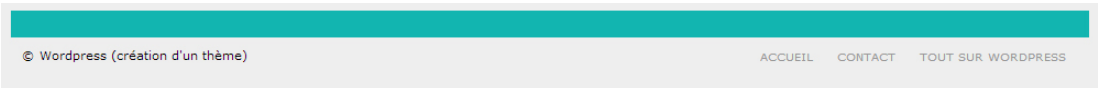
Composer le menu

Consiste à ajouter des rubriques aux menus. Ces rubriques sont soit des pages, des liens ou des catégories.



Ajout de rubriques aux menus

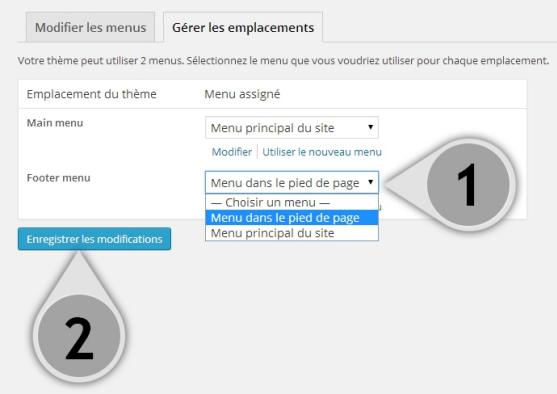
1. Sélectionner le menu à modifier, puis cliquer sur le bouton « Sélectionner ».
2. Ajouter des pages, liens ou catégories en les sélectionnant et en cliquant sur le boutons « Ajouter au menu ».
3. Les options et la position des rubriques peuvent être modifiées.
4. Cliquer sur le bouton « Enregistrer le menu ».



Le résultat dans le pied de page du site

Gérer les emplacements

Les assignations des menus aux positions du thème peuvent être modifiées en allant dans l'onglet « Gérer les emplacements » de l'administration du CMS.



Gestion des emplacements des menus dans les zones réservées du thème

1. Définir dans la liste déroulante le menu de l'administration qui doit apparaître dans la position indiquée.
2. Cliquer sur le bouton « Enregister les modifications ».

La gestion des Widgets

Comme pour les menus, les Widgets peuvent être organisés et disposés dans plusieurs zones du thème. L'appel à la fonction `dynamic_sidebar()` permet de définir une zone dans le thème qui permet la gestion des Widgets dans le site. Il est également possible d'obtenir plusieurs zones dans le même thème et d'afficher les Widgets à plusieurs endroits.

1. **Indiquer l'emplacement dynamique de la zone contenant les Widgets dans le thème**

   Les zones pourront être indiviuellement appelées dans le thème en introduisant la fonction `dynamic_sidebar()` à l'emplacement souhaité. Seul le nom de la zone doit être indiqué comme paramètre à la fonction. Grâce à ce nom une liaison avec l'administration du CMS pourra être établie à l'étape suivante.

```
<aside>
  <?php  dynamic_sidebar( 'zone-1' ); // Appel d'une première zone ?>
</aside>


<aside>
  <?php  dynamic_sidebar( 'zone-2' ); // Appel d'une première zone ?>
</aside>
```

Nom de la zone

2. Appel à la fonction PHP de l'administration des menus

functions.php

Dans le fichier « **functions.php** » seront transmises les positions des zones du thème par la fonction `register_sidebar()` qui rend possible l'administration de ces zones dans le CMS. Comme pour les menus, une dénomination devra être définie pour chaque zone. Celle-ci sera utilisée par l'administration du CMS pour identifier les zones du thème pouvant accueillir des Widgets.

La fonction `register_sidebar()` propose de nombreux paramètres. Pour en savoir plus une visite à la page (http://codex.wordpress.org/Function_Reference/register_sidebar) s'impose.

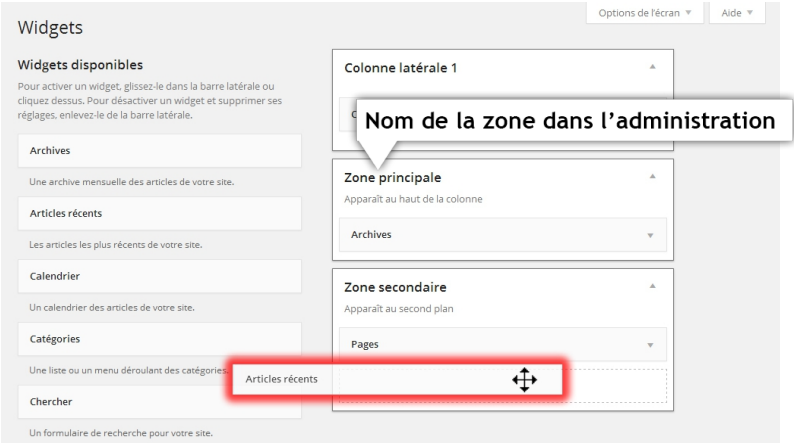
```php
<?php
function my_sidebars_init(){
   register_sidebar(
          array(
                 'name' => __( 'Zone principale', 'montheme' ),
                 'id' => 'zone-1'
          )
   );
   register_sidebar(
          array(
                 'name' => __( 'Zone secondaire', 'montheme' ),
                 'id' => 'zone-2',
                 'description' => __( 'Apparaît au second plan', 'montheme' )
          )
   );
}

add_action( 'widgets_init', 'my_sidebars_init' );
?>
```

Nom de la zone dans l'administration

Nom de la zone

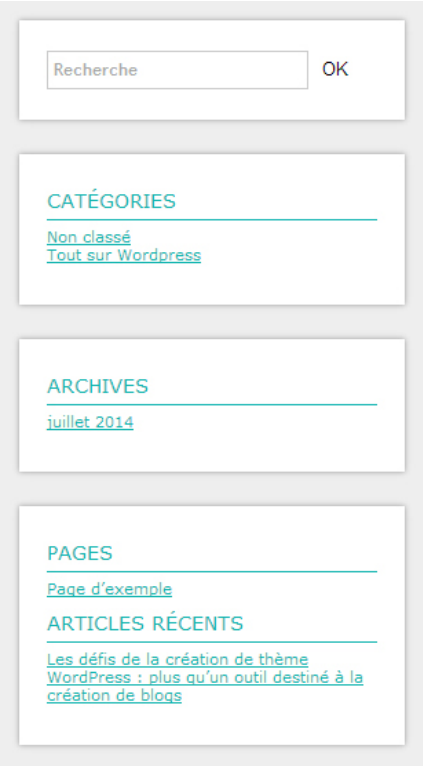
3. Gérer les zones depuis l'administration

Les zones citées dans la fonction `register_sidebar()` sont dorénavant présentes dans l'administration du CMS. Ce qui permet l'assignation de Widgets.



Les 3 méthodes d'insertion de Widgets

La gestion des modules peut se faire de plusieurs façons selon ce qui a déjà été indiqué dans ce document.
Cette colonne de plusieurs zones illustre la mixité possible des méthodes.



Statique (imposé dans le code - « en dur »)

```
<form method="get" id="searchform" action="<?php bloginfo('home'); ?>/">
<input type="text" value="<?php the_search_query(); ?>" name="s" id="s" />
     <button>OK</button>
</form>
```

Appel par fonction Wordpress en PHP

```
<?php get_search_form(); ?>
```

Appel dynamique

```
<aside>
<?php  dynamic_sidebar( 'zone-1' ); ?>
</aside>
```

Solution combinée (dynamique et statique-fonction)

```
<?php if( !function_exists('dynamic_sidebar') || !dynamic_sidebar() ): ?>
          // ...
<?php endif; ?>
```

Sous conditions

Il est possible de vérifier si une zone contient des Widgets. Une zone laissée vide de Widgets depuis la zone d'administration du CMS peut ainsi ne pas apparaître, il en va de même pour les éléments HTML qu'elle contient.
La fonction `is_active_sidebar()` vérifie si la zone contient des Widgets ou pas.

```
<?php if ( is_active_sidebar( 'zone-1' ) ) : ?>
        <article>
        <?php dynamic_sidebar( 'zone-1' ); ?>
        </article>
<?php endif; ?>
```

Plusieurs fichiers sidebars

Il est possible d'ajouter plusieurs fichiers au thème afin que celui-ci dispose de plusieurs sidebars. Pour cela, il suffit d'utiliser la fonction PHP de Wordpress `get_sidebar()` qui fera référence au suffixe employé dans le nom du fichier indiqué entre les parenthèses de la fonction.

`<?php get_sidebar(); ?>`	Fait appel au fichier sidebar.php du thème
`<?php get_sidebar( 'main' ); ?>`	Fait appel au fichier sidebar-main.php

Augmenter les fonctionnalités du thème

D'avoir réussi à rendre le thème complètement dynamique est une très bonne chose mais qui peut s'avérer insuffisant pour les cas particuliers qui impliquent de modifier le traitement du contenu, l'apparence ou la mise en page de certaines pages du site. WordPress propose différentes solutions pour ces cas.

Les conditions PHP *is_nomdelafonction*()

WordPress propose la possibilité de vérifier si une information existe dans la page ou d'identifier l'emplacement de l'utilisateur dans le site grâce à des conditions prédéfinies. Ces conditions peuvent être intégrées dans n'importe quel fichier PHP du thème qui dispose de code HTML. Elles s'emploient comme dans l'exemple suivant.

```
<?php if( is_category( '9' )) : ?>
      <h3>La catégorie spéciale</h3>
<?php else : ?>
       <h3>La catégorie normale</h3>
<?php endif; ?>
```

La condition-ci indique que dans le cas où le contenu de la page consultée fait partie de la catégorie dont l'identifiant dans la base de données est '9', le titre sera « La catégorie spéciale ». Pour tous les autres contenus le titre sera « La catégorie normale ».

Dans l'exemple, il est également possible d'utiliser le nom de la catégorie au lieu de l'identifiant comme paramètre à la fonction.

Il existe une grande quantité de conditions PHP que WordPress propose et qui permettent de modifier l'affichage des informations dans le thème. Voici quelques conditions intéressantes. Pour une liste complète, consulter la documentation de WordPress (http://codex.wordpress.org/Conditional_Tags).

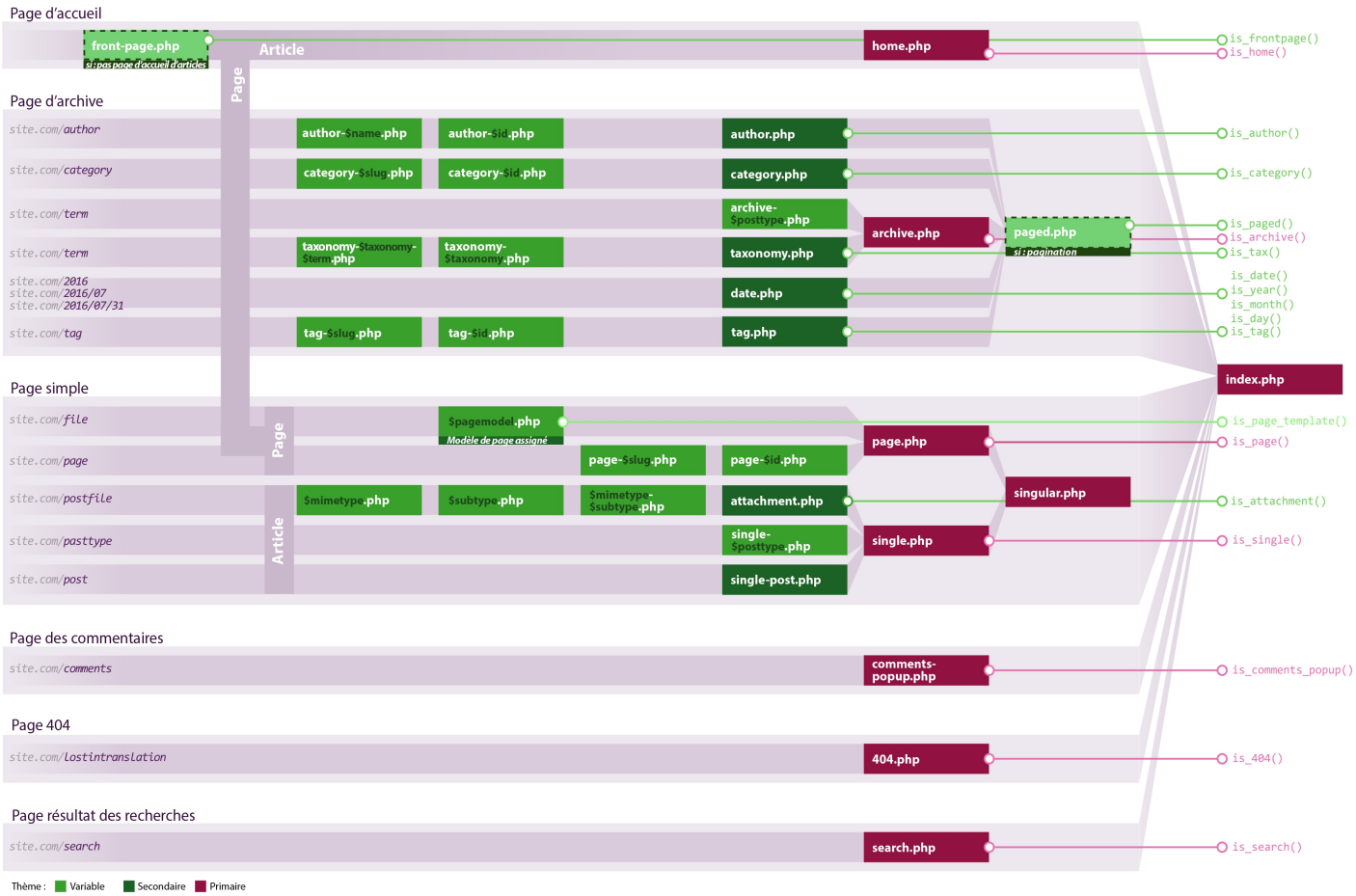
`is_home()`	Détecte si la page consultée par le visiteur est la **page d'accueil** du site.
`is_single()`	Détecte si la page consultée par le visiteur est celle d'**un article**.
`is_page()`	Détecte si la page consultée par le visiteur est celle d'**une page spécifique** (telle qu'indiquée par la valeur transmise à la fonction).
`is_author()`	Détecte si la page consultée par le visiteur est celle d'**un auteur spécifique** (telle qu'indiquée par la valeur transmise à la fonction).
`is_archive()`	Détecte si la page consultée par le visiteur est celle en lien avec **une archive**.
`is_404()`	Détecte si la page consultée par le visiteur **n'existe pas ou plus**.
`is_user_logged_in()`	Détecte si **le visiteur** qui consulte le site **est enregistré**.
`is_dynamic_sidebar()`	Détecte si le thème permet la **gestion des Widgets depuis la zone d'administration** du CMS.

Les sous-pages

Par défaut, c'est le fichier du thème « index.php » qui est consulté et affiché lorsqu'un utilisateur consulte les pages du site. Ceci laisse supposer que toutes les pages du site auront le même modèle de mise en page... WordPress offre la possibilité de définir des aspects graphiques spécifiques pour les différentes portions d'un site grâce aux sous-pages.

Des fichiers portant des noms prévus par WordPress **peuvent remplacer la page « index.php »** dans plusieurs cas prédéfinis par le CMS. Il est ainsi possible d'ajouter au thème un fichier « home.php » afin de définir une mise en page, l'organisation du contenu et la mise en forme en indiquant du HTML et CSS spécifique pour cette page. Du moment où le fichier existe dans le répertoire du thème, il aura le dessus sur la page « index.php » et la remplacera automatiquement à l'affichage de la page d'accueil du site.

Le schéma suivant illustre la priorité des fichiers qui peuvent être ajoutés comme sous-pages et ainsi remplacer le fichier « index.php ».



Comme ce schéma l'indique, il est possible de créer un fichier PHP sous la forme d'une sous-page qui porte le nom ou l'identifiant d'une page telle que spécifiée dans l'administration du CMS lors de sa création. Résultat, il est possible de réserver un thème pour une page spécifique en disposant d'un fichier portant le nom « page-1.php » (dans le cas d'un identifiant. Cette pratique se décline pour les auteurs, les catégories, les étiquettes, les pages et les articles.

Les modèles de page

Les modèles de page sont des fichiers qui fonctionnent comme les sous-pages. Ils offrent la possibilité de définir une apparence et une organisation spécifiques aux pages, et peuvent en plus être assignés depuis la zone d'administration du CMS.

Fabrication d'un modèle de page :

1. **Créer le fichier PHP modèle**

   Le fichier peut porter n'importe quel nom mais doit être logé dans le dossier du thème. Il peut suffire de créer une copie du fichier « index.php » du thème et d'effectuer les modifications de mise en page et de mise en forme.

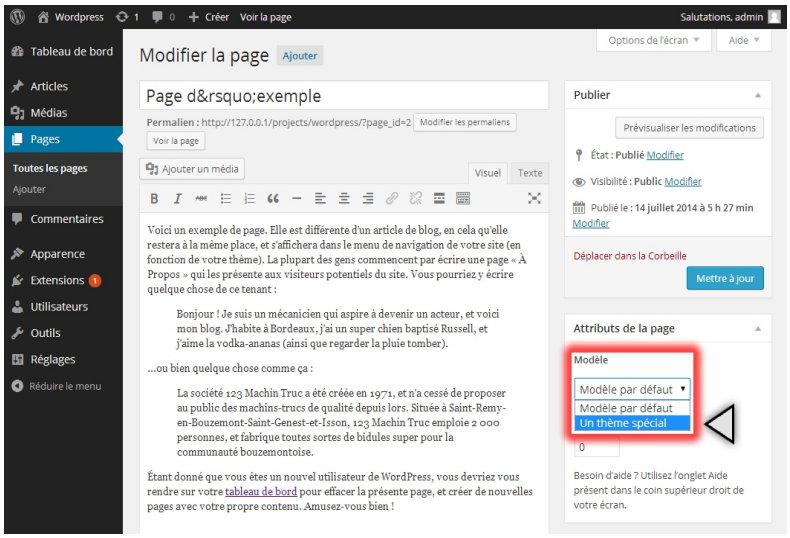
2. **Ajouter le code PHP**

   Il s'agit d'un commentaire PHP qui doit être inséré tout au début du fichier. Ce commentaire permet d'indiquer le nom du modèle et sera reconnu par l'administration du CMS.

```
<?php
/*
Template Name: Un thème spécial
*/
?>
```

3. **Assigner le modèle à une page**

   L'assignation est réservée aux pages et se fait depuis la zone d'administration du CMS. Une liste déroulante est accessible depuis l'espace d'édition de pages.



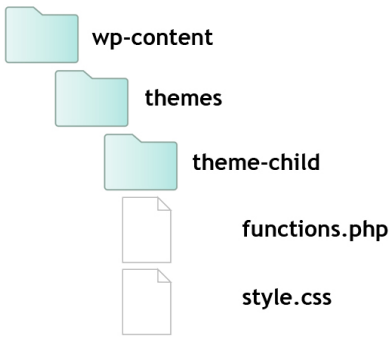
Les thèmes enfants (sous-thèmes)

Créer un thème peut s'avérer fastidieux. Utiliser un des très nombreux thèmes existants peut parfois paraître judicieux pourvu qu'il convienne aux besoins du site. Le paramétrage depuis l'administration du CMS de thèmes installés peut parfois (rarement) suffire. Il est cependant souvent nécessaire d'intervenir dans le code des fichiers du thème afin d'effectuer des changements ciblés. Au-delà des difficultés que représente l'opération, modifier les fichiers d'un thème comporte le risque qu'un jour il faille mettre à jour le thème ce qui signifie que la totalité ou une partie des fichiers du thème seront remplacés et les modifications effectuées supprimées par la mise à jour.

La création d'un sous-thème permet d'éviter que les modifications effectuées d'un thème puisse être affectées lors d'une mise à jour. Il est donc possible de modifier un thème sans avoir à intervenir dans les fichiers propres au thème. La création d'un thème enfant s'avère en plus un moyen efficace d'intervention qui permet de gagner du temps et de disposer de fichiers contenant uniquement les modifications souhaitées, donc faciles à entretenir ou corriger.

La création d'un thème enfant se fait en créant un dossier dans **[wp-content/themes]** portant le même nom que le thème à modifier mais en y ajoutant **[-child]** à la fin. Ce dossier doit contenir les fichiers **[style.css]** et [functions.php].

1. Créer dans [**wp-content/themes**], un répertoire qui contiendra le thème enfant. Donner le nom du thème à modifier et ajouter **[-child]** à la fin.
2. Créer dans ce répertoire les fichiers **functions.php** et **style.css**.
3. Les informations suivantes sont à indiquer dans le fichier **style.css**




```
/*
Theme Name:   Thème enfant
Theme URI:    http://mon-site-theme.com/
Description:  Description de ce thème enfant
Author:       Prénom Nom
Author URI:   http://mon-site.com
Template:     twentyfifteen
Version:      1.0.0
License:      GNU General Public License v2 or later
License URI:  http://www.gnu.org/licenses/gpl-2.0.html
Tags:         mot-clé, mot-clé2, mot-clé3
Text Domain:  twenty-fifteen-child
*/
```

Note **:** Le nom figurant à Template, doit être celui du répertoire du thème de référence (parent).

4. Dans le fichier functions.php il faudra préciser l'ordre de chargement de la feuille de styles du thème en ajoutant le code suivant :

```php
<?php
	add_action( 'wp_enqueue_scripts', 'theme_enqueue_style' );
	function theme_enqueue_style(){
	    wp_enqueue_style( 'parent-style', get_template_directory_uri().'/style.css' );
	}
?>
```

Note **:** En accédant à la rubrique 'Apparence' dans l'administration du site, on peut constater que le thème est reconnu par le CMS. Il peut donc être activé.

Remplacer un fichier du thème parent

Du moment où le thème enfant est créé, il est possible de remplacer (suppléer) n'importe quel fichier du thème parent. Pour cela, il suffit de créer un fichier portant le même nom que le fichier à remplacer et de recoder le fichier selon les modifications souhaitées.

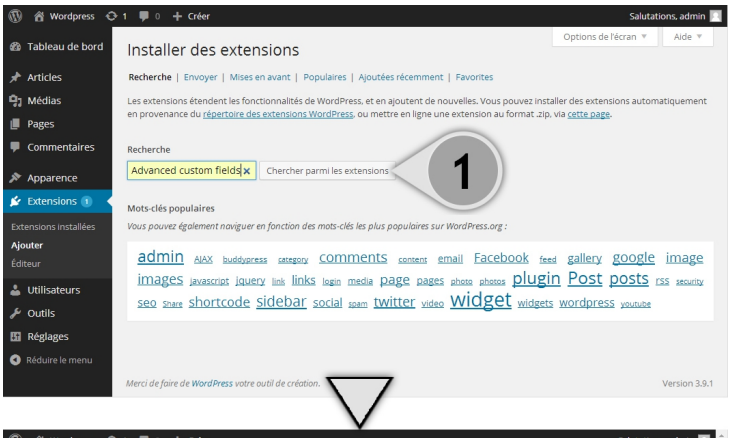
Il est également possible de compléter les fichiers du thème en ajoutant des sous-thèmes ou des modèles de page.

Les extensions

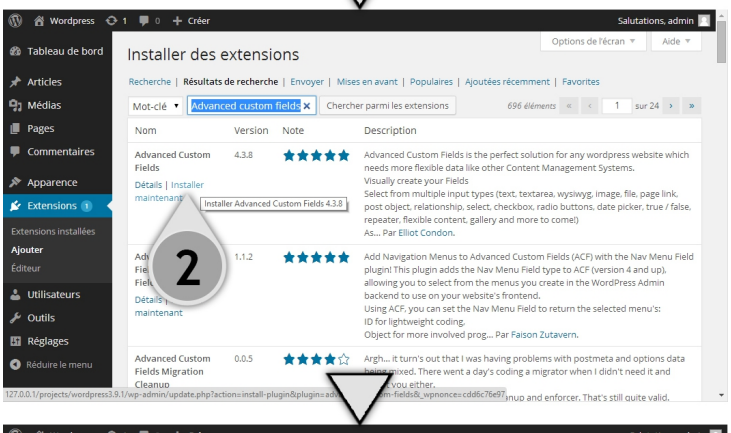
Les extensions ajoutent des fonctionnalités au CMS. Il existe des extensions pour tous les usages qui peuvent être envisagés avec ce CMS (ou à peu près). Malgré leur nombre, trouver celui qui correspond aux besoins spécifiques d'un projet peut relever d'un exploit. Par conséquent, les exigences sont parfois revues ou des développements complémetaires peuvent s'avérer nécessaires.

Installer une extension

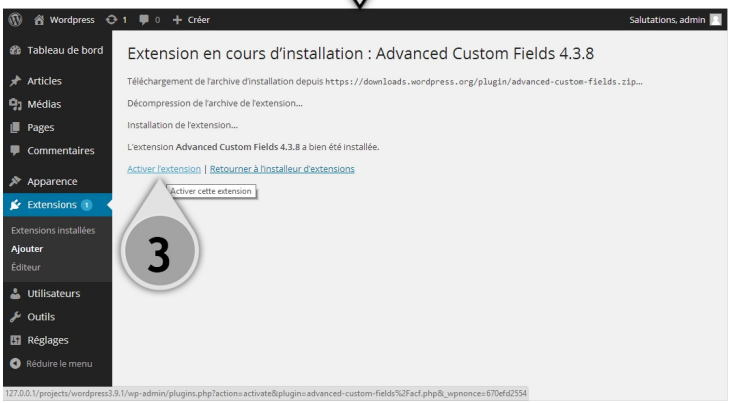
L'installation consiste a télécharger depuis la zone d'administration du CMS les extensions souhaitées. L'outil d'installation offre la possibilité de trouver des extensions à travers les catégories d'extensions ou un agent de recherche.



1. Recherche d'une extension par son nom avec l'agent de recherche. Cliquer sur « Chercher parmi les extensions ».



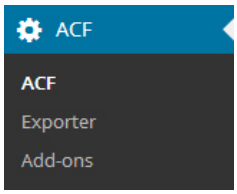
2. Effectuer le téléchargement puis l'installation en cliquant sur le lien « Installer maintenant ».

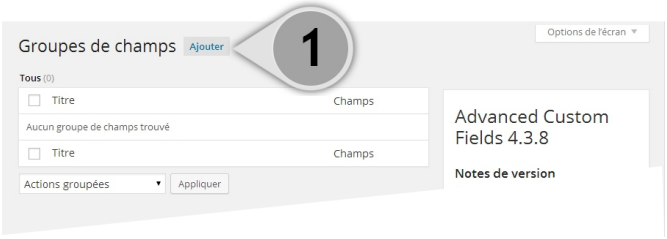


3. Rendre opérationnel l'extension en cliquant sur le lien « Activer maintenant ».

L'extension « Advance Custom Fields »

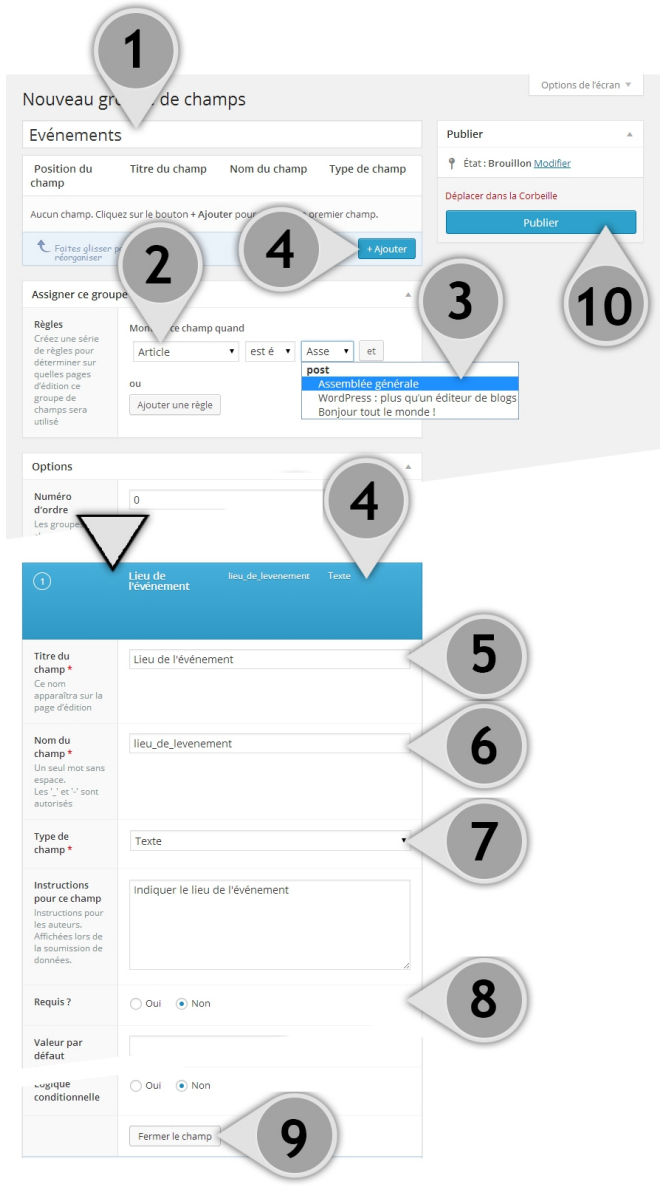
L'extension « Advance Custom Fields » apporte une modularité supplémentaire au CMS en offrant la possibilité d'ajouter des champs supplémentaires aux articles et autres contenus du CMS. Cette extension ajoute une rubrique dans le menu de l'administration du CMS qui permet de le paramétrer. Il sera également question d'ajouter dans le thème des fonctions PHP spécfiques à cette extension pour afficher les contenus supplémentaires des articles.





1. Créer un groupe de champs qui sera assigné à un élément de l'administration en cliquant sur le bouton « Ajouter ».

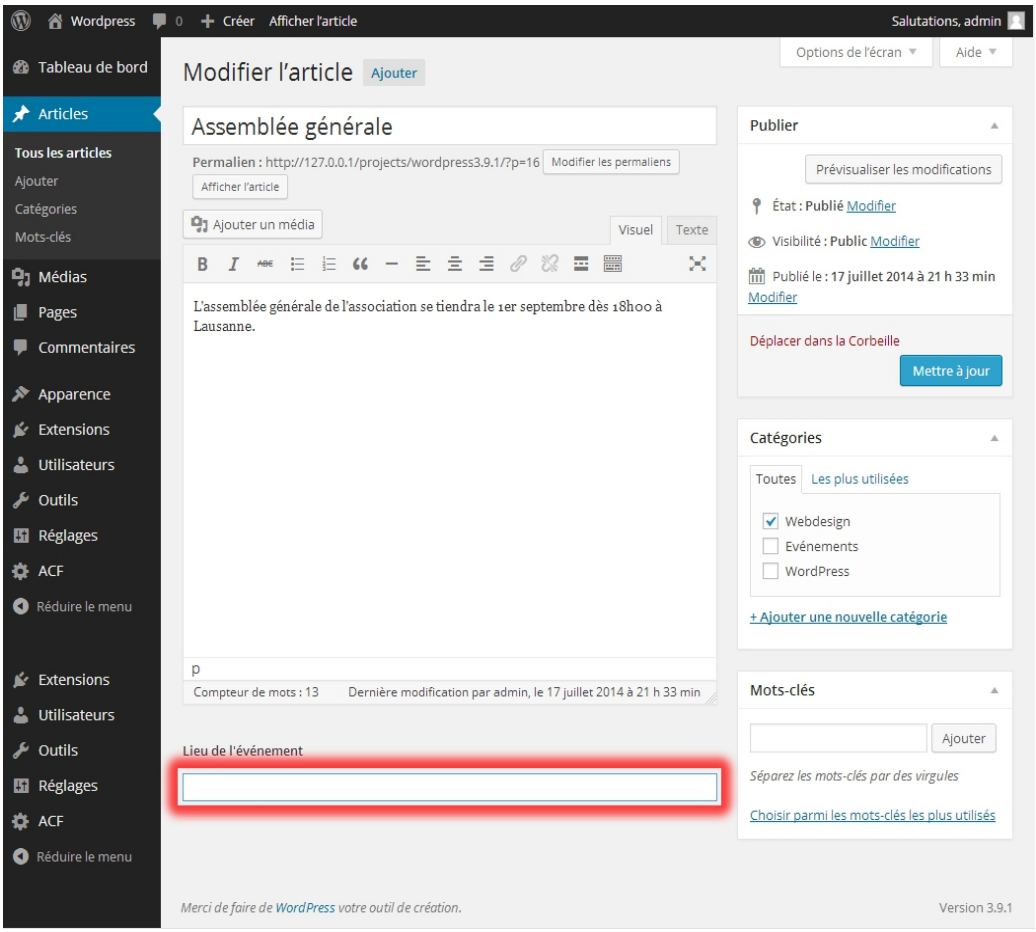
Créer des champs pour des éléments de la zone d'administration



1. Indiquer le nom du groupe de champs.
2. Définir l'élément de la zone d'administration à qui le ou les champ(s) seront ajouté(s). Dans notre exemple, un article.
3. En complément, définir l'élément spécifique à qui ce groupe de champs sera assigné.
4. Créer un champ en cliquant sur le bouton « Ajouter ». Cette opération fait apparaître une nouvelle partie du formulaire qui permettra de paramétrer le champ à créer.
5. Indiquer le nom du champ à créer.
6. Indiquer l'alias du champ qui servira à l'introduire dans le code du thème. Ce nom se crée automatiquement sur la base de la formulation indiquée lors de la nomination du champ à l'étape précédente.
7. Indiquer le type de champ. En l'occurrence un champ de type texte.
8. Indiquer si le champ est obligatoire.
9. Une fois le paramétrage terminé cliquer sur le bouton « Fermer le champ ».
10. Générer le groupe et son assignation en cliquant sur le bouton « Publier ».

Ajouter des contenus supplémentaires aux articles

Une fois le groupe créé, le ou les champs créés seront disposés dans le formulaire des éléments de l'administration du site. Il sera ainsi possible d'ajouter des informations supplémentaires qui seront publiés sur le site.



Insérer les fonctions PHP de l'extention au thème

Afin d'afficher les informations des champs créés grâce à l'extension « Advanced Custom Fields » la fonction `the_field()` sera à ajouter dans le thème dans le ou les fichiers concernés. Il est nécessaire, que cette fonction figure dans la boucle PHP de WordPress qui génère les contenus dans le cas donné, puisqu'il est assigné à un article.

La fonction `get_field()` permet de vérifier si le champs existe lors de l'exploitation de la boucle. Elle devra être insérée dans une condition afin d'éviter que les articles qui ne disposent pas de ce champ génère une erreur à l'affichage.

```
<?php if( get_field( "lieu_de_levenement" ) ): ?>

        <p><?php the_field( "lieu_de_levenement" ); ?></p>

<?php endif; ?>
```

Les fonctions PHP supplémentaires utiles

Wordpress dispose de multiples fonctions pour des usages très variés, que l'on peut intégrer afin d'augmenter la flexibilité du thème ou de déclencher tout simplement une opération très spécifique. Voici deux d'entre elles qui sont souvent rencontrées et qui s'avèrent très utiles.

Les requêtes d'articles

Il peut s'avérer pratique de disposer dans une catégorie des articles dans le but de les afficher à un endroit spécifique dans le site, sur toutes les pages ou encore directement sur la page d'accueil. Les requêtes sélectionnent dans la base de données les articles à afficher sur la page. En précisant ces requêtes on peut conditionner le résultat à l'affichage.

Voici la boucle provenant de la requête principale qui génère automatiquement le contenu de la page courante (que ce soit pour des articles ou des pages).

La fameuse boucle...

```
<?php if(have_posts()) : ?>
	<?php while(have_posts()) : the_post(); ?>
		<article><?php the_content(); ?></article>
	<?php endwhile; ?>
<?php endif; ?>
```

Dans le cas où il est question de modifier la requête principale, la fonction `get_posts()` peut être utile. Elle permet d'ajouter des conditions à la requête pour en modifier le résultat.

Dans l'exemple, récupère au maximum 5 articles d'une catégorie dont l'identifiant est '1'.

La fonction wp_reset_postdata() est indiquée en fin de boucle pour supprimer les valeurs contenues dans la variable $post. Ainsi, une seconde requête peut être définie sans s'inquiéter de la présence des valeurs de la première requête.

```
<?php
$args = array( 'posts_per_page' => 5, 'category' => 1 );
$myposts = get_posts( $args );
	foreach ( $myposts as $post ) : setup_postdata( $post ); ?>
		<li>
			<a href="<?php the_permalink(); ?>">
			<?php the_title(); ?></a>
		</li>
		 <?php
	endforeach;
wp_reset_postdata();
?>
```

Pour en savoir plus, consulter la page (http://codex.wordpress.org/Template_Tags/get_posts).

Dans le cas où il est question de composer une requête de toute pièce, l'appel à l'objet `WP_Query()` sera nécessaire. Il permet de récupérer des articles sans tenir compte de la requête principale de la page courante.

```php
<?php
$my_query = new WP_Query( 'cat=1,2,5' );
if( $my_query->have_posts() ){
	while( $my_query->have_posts()){
		$my_query->the_post(); ?>
		<article><?php the_content(); ?></article>
	<?php
	}
}
wp_reset_postdata();
?>
```

Dans l'exemple, récupère les articles des catégories dont l'identifiant est '1', '2' ou '5'.

Pour en savoir plus, consulter la page (http://codex.wordpress.org/Class_Reference/WP_Query).

Introduire des « shortcodes » directement dans le thème

Wordpress permet d'introduire certains contenus tels que des extensions ou les médias dans des articles ou des pages par l'entremise de codes spécifiques au CMS, appelés « shortcode ». Il s'agit d'un moyen plutôt simple à utiliser même pour un administrateur de site qui ne connaît très peu de choses à la programmation.

Un « shortcode » à cette apparence :

```
[gallery id="123" size="medium"]
```

Il indique le rappelle le nom d'un outil auquel des attributs lui sont ajoutés. Dans l'exemple ci-dessus, il s'agit d'un « shotrcode » qui fait référence à une extension du type galerie pour afficher le diaporama portant l'identifiant « 123 » à une taille moyenne.

Cet outil, pratique peut également être inséré dans un thème. Pour cela, il faut utiliser la fonction do_shortcode() à l'endroit où devrait apparaître l'outil appelé.

```php
<?php
	do_shortcode( '[gallery id="123" size="medium"]' );
?>
```

Pour en savoir plus consulter la page (https://codex.wordpress.org/Shortcode_API).

L'ajout de fonctions PHP au thème

Le fichier « functions.php » peut contenir des fonctions spécifiques au thème en développement. Ces fonctions entièrement conçues en PHP offrira des fonctionnalités que l'auteur du thème aura envisagé et seront ajoutées aux fonctions PHP de WordPress. Pour cela il existe la fonction `add_action()` qui permet d'ajouter à une fonction existante de nouvelles fonctionnalités.

```php
<?php
function email_a_friend() {
        $friends = 'friend@example.com;
        $title = 'Mise à jour';
        $message = 'Du nouveau contenu a été ajouté';
        wp_mail( $friends, $title, $ message );
}
add_action( 'publish_post', 'email_a_friend' );
?>
```

Ce code se trouvera dans le fichier « functions.php ».

la fonction email_a_friend() a été conçue de toute pièce. Elle sera appelée et exécutée au moment où la fonction PHP de WordPress publish_post() sera également utilisée.

Pour en savoir plus consulter la page (http://codex.wordpress.org/Function_Reference/add_action).

Structure et arborescence d'un thème

Le processus de création d'un thème sur mesure implique de comprendre les moyens de structurer les contenus avec Wordpress et de connaître ses ressources leurs possibilités ainsi que leurs limites.

Ressource 1 : Distinction entre « Page », « Article » et « Catégorie »

Deux facteurs sont déterminants pour distinguer si une page du site doit être conçue dans l'administration de CMS comme une « Page » ou un « Article » et dans ce cas si elle devrait être classée dans une « Catégorie » :

- **L'édition d'une page et son contenu**

  Wordpress considère qu'une « Page » n'est qu'un contenu statique et unique. Elle ne se combine pas avec autre chose. Ce qui est toutefois le cas des « Articles » qui peuvent être groupés dans une seule page. Les événements proposés dans un site sont des « Articles » dans le CMS Wordpress.

- **Le classement dans les menus du site**

  Wordpress offre la possibilité de composer les rubriques d'un menu d'un site en accédant directement à une « Page », un « Article », une « Catégorie » (regroupant des articles) ou un lien externe. Il est ainsi possible de grouper dans une même page plusieurs articles. Pour organiser le classement et la récupération de plusieurs articles dans une même page, il s'avère utile de définir des catégories. A noter que cette technique est également utilisé dans lors de la création de plusieurs boucles dans une même page.

Limites :

Certains cas de figure ne peuvent être résolus uniquement avec cette organisation car il y a des contenus supplémentaires dans certains cas à ajouter. Cela se fera par :

- l'appel d'autres boucles,
- l'utilisation d'une extension appelée par un « shortcode »
- l'utilisation d'une extension comme ACF, offrant la possibilité d'ajouter des champs supplémentaires aux articles ou aux pages,
- l'appel d'une fonction PHP Wordpress

Ressource 2 : Plusieurs boucles dans une page

La boucle d'origine utilisée dans un thème permet d'afficher les contenus de la page courante. Il peut s'avérer utile d'ajouter d'autres contenus dans une page. Par exemple, une page d'accueil affichant des actualités qui sont en fait des « Articles » groupés dans une « Catégorie ».

Pour qu'une nouvelle boucle puisse se faire, une requête doit être établie à l'aide de l'objet `WP_Query()` qui permet d'indiquer les conditions de sélection des articles ou page *(page 35)*.

Ressource 3 : Adapter le comportement d'une fonction PHP Wordpress

Les fonctions PHP conçues par Wordpress pour la récupération de contenus peuvent être paramétrées pour :

- conditionner les contenus à afficher
- modifier l'organisation de la structure HTML résultante
- préciser le mode d'édition dans l'administration du CMS

Pour connaître les dispositions possibles, il est impératif de se référer à la documentation officielle de Wordpress (https://codex.wordpress.org) qui présente toutes les fonctions et leurs particularités.
Une fonction dont le paramétrage est pratique, est `wp_nav_menu()`.

Fonction	HTML (généré)

`wp_nav_menu()`	`<div class="menu">`
	`  <ul id="menu-menu-1" class="menu">`
	`  <li id="menu-item-5" class="menu-item menu-item-type-post_type menu-item-object-page menu-item-5">`
	`  <a href="">Page d'exemple</a>`
	`  </li>`
	`  </ul>`
	`</div>`

`wp_nav_menu(`
`  array(`
`  'theme_location'=>'menu-pr',`
`  'container'=>''`
`  )`
`)`
	`<ul id="menu-menu-1" class="menu">`
	`  <li id="menu-item-5" class="menu-item menu-item-type-post_type menu-item-object-page menu-item-5">`
	`  <a href="">Page d'exemple</a>`
	`  </li>`
	`</ul>`

`wp_nav_menu(`
`  array(`
`  'theme_location'=>'menu-pr',`
`  'container'=>'',`
`  'menu_class'=>'menu-pr'`
`  )`
`)`
	`<ul id="menu-menu-1" class="menu-pr">`
	`  <li id="menu-item-5" class="menu-item menu-item-type-post_type menu-item-object-page menu-item-5">`
	`  <a href="">Page d'exemple</a>`
	`  </li>`
	`</ul>`

`wp_nav_menu(`
`  array(`
`  'theme_location'=>'menu-pr',`
`  'container'=>'',`
`  'items_wrap'=>'%3$s'`
`  )`
`)`
	`<li id="menu-item-5" class="menu-item menu-item-type-post_type menu-item-object-page menu-item-5">`
	`  <a href="">Page d'exemple</a>`
	`</li>`

Limites :

Certains paramétrages ne sont pas proposés par une fonction. Dans ce cas, il est peut s'avérer nécessaire de figurer une solution avec les feuilles de style CSS ou de recomposer un élément en HTML plutôt que d'utiliser la fonction qui le génère *(page 25)*. La fonction `get_search_form()` qui génère un formulaire de recherche ne permet pas de modifier le nom inscrit sur le bouton de recherche. Une solution conçue en HTML peut dans ce cas être plus flexible pour cette raison.

Ressource 4 : Les « schortcodes »

Les « shortcodes » sont un moyen d'introduire du contenu dans un article, page ou directement dans un thème *(page 36)*. Le contenu appelé par un « shortcode » proviennent très souvent d'extensions utilisant cette ressource pour proposer à l'administrateur du site une solution pour introduire du contenu généré par l'extension.

Il est également possible de générer un « shortcode » avec du contenu en créant une fonction dans le fichier « functions.php » comme dans l'exemple que suit qui permet d'introduire un menu via un « shortcode » conçu sur mesure `[shortCode_News]` :

```
function scNews(){
     echo '<ul class="article page">';
          $myquery = new WP_Query(
                        array( 'category_name'=>'news', 'posts_per_page'=>-1, 'order'=> 'DESC' )
                        );
          if( $myquery->have_posts() ):
                while( $myquery->have_posts() ): $myquery->the_post();
                    echo '<li><a href="'; the_permalink(); echo '">'; the_title(); echo '</a></li>';
                endwhile;
           endif;
     wp_reset_postdata();
     echo '</ul>';
}
add_shortcode('shortCode_News', 'scNews');
```

Ressource 5 : ACF

L'extension « **Advanced Custom Fields** » *(page 33)* permet d'ajouter des champs le formulaire d'édition de « Pages » ou certains « Articles ». Ces contenus supplémentaires permettent au thème d'acquérir une flexibilité intéressante.

Il est possible d'indiquer des conditions particulières pour que les champs n'apparaissent que pour certaines « Pages », « Articles » ou en fonction d'une « Catégorie » d'articles.

Ressource 6 : Les sous-pages et les thèmes de page

La spécificité d'une mise en page fait que l'utilisation d'une sous-page *(page 28)* peut s'avérer nécessaire. Cette solution est fréquemment utilisée pour la page d'accueil avec la création d'un fichier « **front-page.php** » ou « **home.php** » qui remplacera d'office « **index.php** » lors de la consultation de la page d'accueil du site.

Cette solution peut s'avérer également utile pour des cas plus ciblé ou spécifique. Par exemple, une page contenant les articles d'une catégorie ayant une mise en page particulière. Un fichier « **category-**

nomdelacategorie.php » peut être créé spécifiquement pour ce cas de figure et sera automatiquement utilisé lors de la consultation de la page.

Des thèmes de page *(page 29)* peuvent également être utilisés pour qu'une mise en page particulière puisse être appliquée lors de l'édition des pages. Cette solution implique la création d'un fichier supplémentaire dans le dossier du thème qui disposera du commentaire PHP `/* Template Name: Nom du thème spécial */` dans le haut du fichier.

Limites :
Plutôt un inconvénient ! Dans le cas d'une sous-page spécifique disposant du nom d'une catégorie, page ou article (par exemple : « category-nomdelacategorie.php »), si l'administrateur réédite ou supprime la catégorie, page ou article en question la sous-page sera inutile.

Ressource 7 : Les conditions Wordpress

Il peut ne pas être nécessaire d'utiliser des sous-pages ou des thèmes de page dans le cas où les particularités d'une mise en page spécifiques sont limitées. Dans ce cas, l'utilisation d'une condition (page 27) Wordpress peut suffire. Elle permet de limiter ou de créer une alternance d'affichage de certains contenus.

Par exemple :

```
if( is_home() ){
	// Contenu à afficher que sur la page d'accueil
}
```

Ressource 8 : Le HTML

Tout contenu ne doit pas absolument être dynamiquement administré depuis l'administration du CMS. Certains contenus (textes, images) peuvent être insérés dans le thème « en dur ». Il s'agit d'un gain de temps non négligeable.

Ressource 9 : Les extensions

Les extensions offrent un large éventail de possibilités dans la gestion de contenus. Que ce soit dans l'augmentation des fonctionnalités de l'administration ou du thème en encore dans la gestion de contenus spécifiques.

Limites :
Les fonctionnalités propres à l'extension peuvent ne pas entièrement correspondre à celles attendues dans le projet. Maîtriser le paramétrage d'une extension peut prendre du temps et s'avérer complexe ou mal adapté.

Développer une extension Wordpress

Tout comme pour la création d'un thème, Wordpress détecte automatiquement les extensions logées dans le répertoire « plugins » du moment où ce répertoire contient un fichier PHP du même nom et que celui-ci dispose d'un commentaire PHP logé au haut du fichier indiquant la description de l'extension. Ce commentaire doit disposer au minimum des indications suivantes.

```
/*
Plugin Name: Mon extension Wordpress
Description: Description de l'extension.
Author: Nom de l'auteur
Version: 1.0
*/
```

Que peut faire une extensions ?

Une extension peut intervenir sur tout ce que Wordpress offre comme fonctionnalités. Il est notamment possible d'intervenir au niveau :

- Modifier ou ajouter de nouvelles fonctionnalités à la zone d'administration.
- Intervenir dans la base de données et ajouter des tables réservées à cette extension ou utiliser celles existantes et interagir avec leurs données
- Créer un outil sur mesure récoltant et traitant des données en fonction d'une utilisation définie
- Ajouter des fonctionnalités et altérer l'apparence ou disposition de l'administration du CMS.

NB : Tout ce qui peut s'appliquer à une extension peut également être utilisé dans un thème. Il est possible d'appliquer ce qui suit à un thème et offrir ainsi des fonctionnalités supplémentaires au CMS associés à ce thème sans installation supplémentaire.

Le développement d'une extension implique, selon les ressources nécessaires, de bien connaître 3 aspects du CMS :

1. **La gestion et l'organisation des contenus**

Il est utile de :

- Connaître la structure de la base de données de Wordpress et de savoir entre autre que :
    - les informations globales du CMS et du site sont stockés dans la table « **wp_options** ». Cette table peut servir à stocker des données utiles aux extensions.
    - les articles, les pages, les rubriques des menus et très souvent les données des extensions sont stockées dans les tables « **wp_posts** » et « **wp_post_meta** »
    - les menus (nom) et les catégories d'articles sont stockés dans les tables « **wp_terms** », « **wp_termmeta** » et « **wp_terme_relationship** ».

    En savoir plus : https://codex.wordpress.org/Database_Description

- Savoir utiliser et concevoir les **shortcodes** ce qui permet d'introduire des contenus provenant d'extensions dans un article ou une page.

    En savoir plus : https://codex.wordpress.org/Shortcode

2. **Le développement et les API Wordpress**

Les API sont les ressources mises à la disposition des développeurs. Chaque API offre une série de fonctionnalités groupées selon une thématique liée à une exploitation spécifique. Elles donnent l'accès aux fonctionnalités des ressources du CMS et facilitent le développement d'extensions et de thèmes.

Dashboard Widgets API : Il permet d'ajouter de nouveaux widgets au tableau de bord administratif.

Database API : Il permet d'utiliser la méthode Wordpress pour accéder aux données de la base de données.

File Header API : Il permet d'extraire des méta-informations (nom, version, auteur, URI , Description, etc.) à partir de ces fichiers.

Filesystem API : Il permet de lire et écrire des fichiers dans le système de fichiers Wordpress de manière sécurisée.

HTTP API : Il standardise les requêtes HTTP de WordPress. L'API gère les cookies, l'encodage et le décodage gzip, le décodage de blocs (si HTTP 1.1) et diverses autres implémentations de protocole HTTP. L'API standardise les demandes, teste chaque méthode avant l'envoi et, selon la configuration du serveur, utilise la méthode approfondie de requête.

Metadata API : Il permet de récupérer et manipuler les métadonnées d'objets WordPress.

Options API : Il permet de stocker des données dans la base de données. L'API facilite la création, l'accès, la mise à jour et la suppression de ces options. Toutes les données sont stockées dans la table « wp_options ».

Plugin API : Il permet de créer des actions et des filtres dans les fonctions et les méthodes d'accrochage de Wordpress.

Quicktags API : Il permet d'inclure des boutons supplémentaires dans l'éditeur Texte (HTML).

Rewrite API : Il permet de gérer les règles de réécriture des URL. Il est utilisé lors de la mise à jour des règles de réécriture, ainsi que pour trouver l'URL d'un article, d'une page, d'une catégorie d'archives, etc.

Settings API : Il permet de créé et géré des formulaires de manière semi-automatique. Il permet de définir des pages de paramètres, des sections dans ces pages et des champs dans les sections.

Shortcode API : Il permet de créer des contenus qui seront intégrés dans les articles et pages par l'entremise de Shortcodes Wordpress.

Theme Modification API : Il dispose de fonctions et des crochets utiles à la modification du thème.

Theme Customization API : Il permet aux développeurs de thèmes d'ajouter des options spécifiques destinées à la personnalisation du thème.

Transients API : Il offre le moyen de stocker dans la base de données les données mises en cache.

Widgets API : Il permet de créer des widgets Wordpress.

XML-RPC WordPress API : Il permet à d'autres systèmes de se connecter et de communiquer avec WordPress en transmettant les données en XML.

En savoir : https://make.wordpress.org/core/handbook/best-practices/core-apis/

3. **Les crochets (hooks), les actions et les filtres**

Les crochets (hooks) sont utiles pour permettre à une extension de s'imbriquer à une séquence d'opératione. C'est-à-dire que les fonctions d'une extension se déclenchent au moment où un filtre ou une action prévue par le CMS est appelé. Il existe près de 2000 actions filtres ou actions. Elles peuvent être intégrées par l'appel de la fonction `add_filter()` et `add_action()`.

Filtres

Un filtre est une fonction que dispose Wordpress et qui peut être altérée par le passage de paramètres. L'exemple suivant modifie l'apparence des titres d'articles en forçant leur imbrication dans la balise HTML `<em>`.

```php
<?php
add_filter( 'the_title', function( $title ){ return '<em>' . $title . '</em>'; } );
?>
```

*La fonction **add_filter()** : https://codex.wordpress.org/Function_Reference/add_filter*
La liste des filtres disponibles : https://codex.wordpress.org/Plugin_API/Filter_Reference

Actions

Permet le déclenchement de nouvelles fonctions au moment de l'appel d'une fonction prévue par le CMS. L'exemple suivant ajoute l'appel d'une fonction qui affiche une balise HTML au moment ou la fonction Wordpress `wp_head()` est exécutée.

```php
<?php
add_action( 'wp_head', function(){ echo '<meta name="author" content="me" />'; } );
?>
```

La fonction add_action() : https://codex.wordpress.org/Function_Reference/add_action
La liste des actions disponibles : https://codex.wordpress.org/Plugin_API/Action_Reference

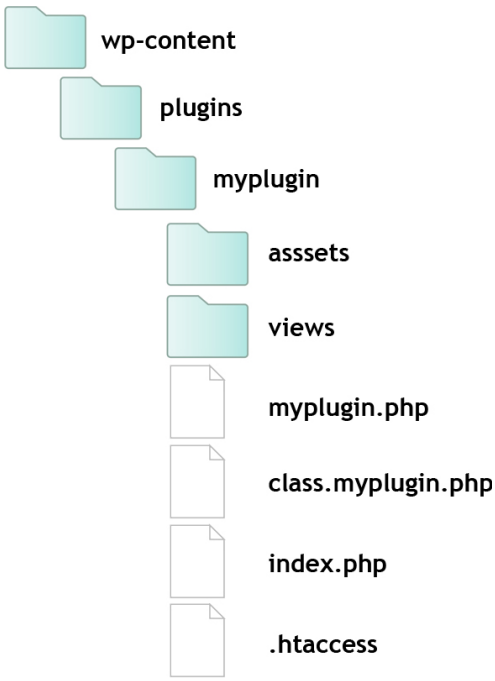
Des actions utiles à l'activation d'une extension et peuvent servir par exemple à ajouter des données dans la base de données ou les retirer lorsque de la désactivation de l'extension.

La structure de base des fichiers de l'extension

Le développement d'une extension implique la mise en place d'outils qui facilitent sa maintenance et les développements futurs. Cette méthode est une réplique de la structure *Model-View-Controller*, répandue dans la gestion de contenus.

Cette structure est établie dans le dossier **[wp-content>plugins]**. Dans l'exemple qui suit l'extension porte le nom « myplugin ». Il contient :

- **assets (dossier)** : Contient les fichiers de mise en forme (CSS), iimages et scripts (JS) des interfaces publics (frontend) de l'extension.
- **view (dossier)** : Contient les fichiers des interfaces d'édition
- **myplugin.php (fichier)** : Fichier principal de l'extension. Il gère le lancement et la séquence des opérations de l'extension. Ce fichier est nécessaire pour la bonne marche de l'extension. Il doit porter le nom du répertoire.
- **class.myplugin.php (fichier)** : Classe principale qui permet dispose des fonctions utiles au fonctionnement de l'extension.
- **index.php (fichier)** : fichier vide qui limite un accès direct au répertoire de l'extension.
- **.htaccess (fichier)** : Conditionne le comportement du serveur à l'égard de l'extension. Ce qui comporte notamment la limitation des accès aux fichiers de l'extension.




```
# Apache 2.2
<IfModule !mod_authz_core.c>
	Order Deny,Allow
	Deny from all
</IfModule>

# Apache 2.4
<IfModule mod_authz_core.c>
	Require all denied
</IfModule>

# Akismet CSS and JS
<FilesMatch "^(myplugin\.js|myplugin\.css)$">
	<IfModule !mod_authz_core.c>
		Allow from all
	</IfModule>

	<IfModule mod_authz_core.c>
		Require all granted
	</IfModule>
</FilesMatch>
```

Quelques règles de sécurité

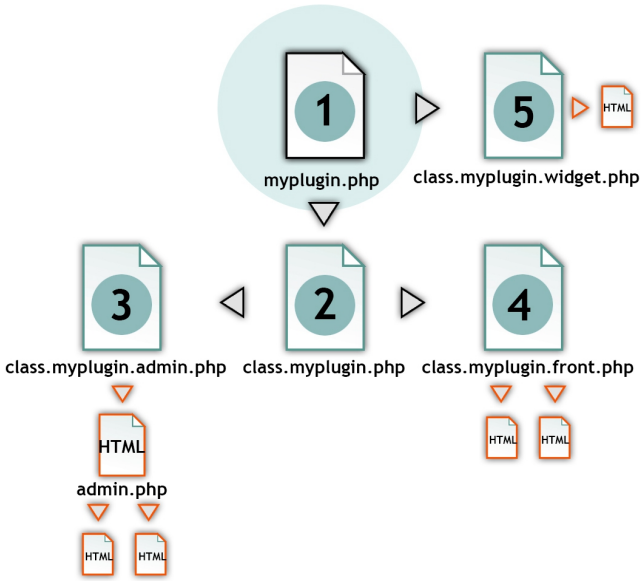
Le développement d'une extension implique d'observer quelques règles quant à l'accès aux fichiers et au traitement des données. Une extension disposant d'une faille de sécurité met en péril un site complet et décrédibilise le travail du développeur :

- disposer d'un fichier index.php vide dans le dossier de l'extension.
- limiter les accès aux fichiers par un contrôle effectué depuis le fichier .haccess
- ajouter une condition dans les premières lignes des fichiers PHP qui permettent de vérifier que c'est bien le CMS qui accède au fichier.
- Utiliser les fonctions prévues par Wordpress pour les interactions avec la base de données et le traitement de données provenant de formulaires.

Structure du code

Le fichier principal (myplugin.php, dans l'exemple de la structure décrite auparavant) est consulté par le CMS dès que l'extension est activée, et ce sans même que l'administrateur interagisse ou configure l'extension.

La structure du code telle qu'établie dans l'extension permet de gérer et moduler le traitement des contenus et les vues (en HTML) générés par l'extension.



1. myplugin.php

Initie l'extension, prévoit les actions à mener à l'activation et à la désactivation de celle-ci. Fait appel à la classes destinée au traitement des contenus et à celle des widgets.

2. class.myplugin.php

Gère les contenus généraux de l'extension (pour ceux qui n'ont pas besoin de vues ou qui se génèrent automatiquement. Délègue aux classes de gestion des contenus d'administration et des contenus du site.

3. class.myplugin.admin.php

Gère les contenus de l'administration et les transmet aux vues correspondantes.

4. class.plugin.front.php

Gère les contenus publics et les transmets aux vues correspondantes.

5. class.myplugin.widget.php

Gère les contenus pour l'administration comme le site public des widgets de l'extension.

myplugin.php

```php
<?php
/*
Plugin Name: Une extension Wordpress
Description: Description de l'extension Wordpress.
Author: Prenom Nom
Version: 1.0
*/

if ( !function_exists( 'add_action' ) )
{
    echo 'Ce fichier ne peut pas être accédé directement.';
    exit;
}
define( 'MY_PLUGIN_PATH', plugin_dir_path( __FILE__ ) );
define( 'MY_PLUGIN_URL', plugin_dir_url( __FILE__ ) );

register_activation_hook( __FILE__, array( 'Myplugin', 'plugin_activation' ) );
register_deactivation_hook( __FILE__, array( 'Myplugin', 'plugin_deactivation' ) );

require_once( MY_PLUGIN_PATH . 'class.myplugin.php' );
require_once( MY_PLUGIN_PATH . 'class.myplugin.admin.php' );
require_once( MY_PLUGIN_PATH . 'class.myplugin.front.php' );
require_once( MY_PLUGIN_PATH . 'class.myplugin.widget.php' );

add_action( 'init', array( 'Myplugin', 'init' ) );

add_action( 'widgets_init', function(){ register_widget( 'My_Plugin_Widget' ); } );
```

Le fichier principal (myplugin.php) associe les fichiers utiles à l'extension, en occurrence les fichiers disposant des classes PHP de l'extension. Il peut ainsi faire appel aux opérations prévues lors de l'activation de l'extension grâce à la fonction `register_activation_hook()` et à la désactivation de l'extension grâce à la fonction `register_deactivation_hook()`. Ces deux fonctions font chacune appel à une méthode inscrite dans la classe `MyPlugin` comme l'indique le paramètre `array('Myplugin', 'plugin_activation')` et `array('Myplugin', 'plugin_deactivation')`.

L'appel de la classe `Myplugin` initie le processus d'initialisation de l'extension en donnant accès aux traitements prévues par la classe. Cela s'effectue par l'appel de la méthode statique `init()` de la classe `Myplugin` par la fonction `add_action( 'init', array( 'Myplugin', 'init' ) );`.

Dans le cas où un widget venait à être créé, l'appel à la classe prévue à cette effet se fait depuis le fichier principal. Cette classe à la particularité de s'étendre à la classe `WP_Widget` de Wordpress et doit être initié au démarrage de l'extension pour surcharger correctement celle-ci par l'action menée par la fonction `add_action( 'widgets_init', function(){ register_widget( 'My_Plugin_Widget' ); } );`.

class.myplugin.php

```php
<?php

class Myplugin{

    private static $initiated = false;

    public static function init()
    {
        if ( ! self::$initiated )
        {
            self::init_hooks();
        }
    }

    private static function init_hooks()
    {
        self::$initiated = true;

        new Myplugin_Admin();

        new Myplugin_Front();
    }

    public static function plugin_activation()
    {

    }

    public static function plugin_deactivation()
    {

    }
}
```

Ce fichier dispose des méthodes `init()`, `plugin_activation()` et `plugin_deactivation()` appelées depuis le fichier principal `myplugin.php`.

Afin d'éviter qu'un appel à plus d'une reprise de cette classe puisse se faire un déclencheur (*trigger*) est prévu par l'entremise de la variable `$initiated` qui changera de valeur dès que la classe aura été appelée un première fois.

A son tour il fait appel aux classes prévues pour la gestion des contenus de l'administration et du site public avec `new Myplugin_Admin();` et `new Myplugin_Front();`. On remarque qu'aucune instanciation de la classe n'est nécessaire. Le constructeur de ces classes feront le nécessaire.

Activation et désactivation de l'extension

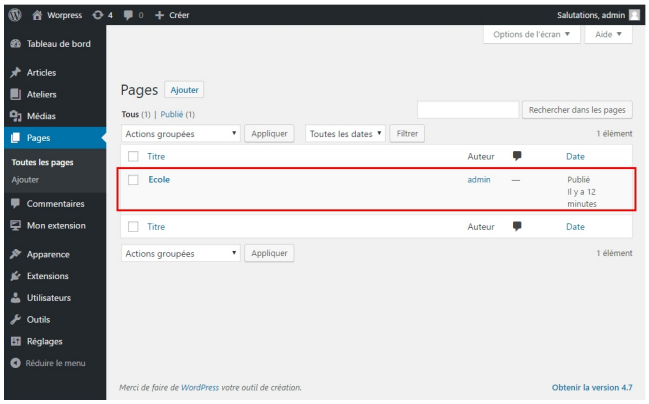
L'activation et la désactivation de l'extension peut donner lieu au déclenchement d'opérations prévues pour le bon fonctionnement de l'extension. Les méthodes `plugin_activation()` et `plugin_deactivation()` proposées précédemment permettent ces opérations. Voici deux exemples d'utilisation concrètes et utiles selon les fonctionnalités prévues.

Création de « *posts* » (article ou page)

L'ajout automatique d'un article ou d'une page à l'aide de la fonction `wp_insert_post()`. L'attribut `'post_type'` est utile pour définir le type de post à insérer. La valeur qui lui est attribuée peut éventuellement être différente de `'post'` (article) ou `'page'` car comme cela sera décrit dans la prochaine section, il est possible de créer ces propres types de « *posts* » et par conséquent leur ajouter des contenus spécifiques.

class.myplugin.php

```
27  [...]
28  public static function plugin_activation()
29  {
30          $post = array(
31              'post_name' => 'ecole',
32              'post_status' => 'publish',
33              'post_title' => 'Ecole',
34              'post_type' => 'page',
35          );
36
37          wp_insert_post( $post );
38  }
39  [...]
```



En savoir plus : https://developer.wordpress.org/reference/functions/wp_insert_post/

Suppression de posts à la désactivation de l'extension se fait à l'aide de la fonction wp_delete_post() selon les conditions transmises comme paramètres.

class.myplugin.php

```
39  [...]
40  public static function plugin_deactivation()
41  {
42          $posts = get_pages( array( 'post_name' => 'ecole') );
43
44          foreach ( $posts as $post )
45          {
46              wp_delete_post( $post->ID, true );
47          }
48  }
49  [...]
```

Ajout de tables dans la base de données

Wordpress prévoit l'interaction avec la base de données grâce à la variable globale `$wpdb` qui instancie la classe du même nom. Elle sera décrite dans une prochaine section. La méthode `query()` permet d'effectuer une requête sécurisée.

class.myplugin.php

```
27  [...]
28  public static function plugin_activation()
29  {
30          global $wpdb;
31
32          $wpdb->query("CREATE TABLE IF NOT EXISTS {$wpdb->prefix}coachs (id_coach INT
    AUTO_INCREMENT PRIMARY KEY, name_coach VARCHAR(100) NOT NULL);");
33  }
34  [...]
```

A la désactivation de l'extension la même méthode peut être employée pour supprimer des tables.

class.myplugin.php

```
34  [...]
35   public static function plugin_deactivation()
36   {
37          global $wpdb;
38
39          $wpdb->query("DROP TABLE IF EXISTS {$wpdb->prefix}coachs;");
40   }
41  [...]
```

La gestion automatisée des contenus avec les types de « *posts* »

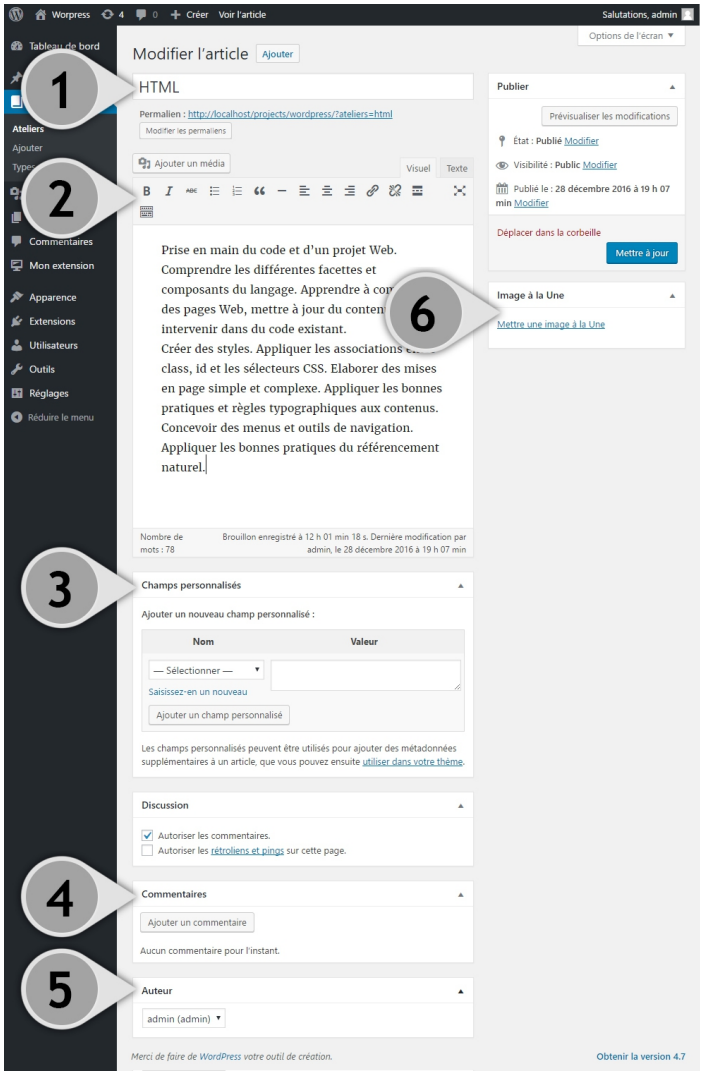
Wordpress propose d'un outil utile pour la création de contenus sans avoir à créer de tables supplémentaires dans la base de données. Cette solution se décline de la gestion des articles et pages, elle permet la création de types de « *posts* » spécifiques à un contenu.

La fonction `register_post_type()` permet la conception d'un type de « *post* ». Elle génère une rubrique dans le menu de l'administration du CMS permettant son accès. Un nom spécifique doit lui être indiqué comme paramètre. Il servira de référence.

class.myplugin.php

```
62  [...]
63  private static function create_posts_type()
64  {
65      register_post_type( 'ateliers',
66          array(
67              'labels' => array(
```

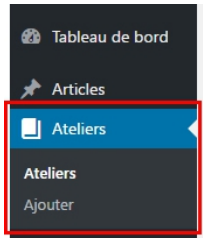
```
68              'name' => __( 'Ateliers' ),
69              'singular_name' => __( 'Ateliers' ),
70          ),
71          'public' => true,
72          'has_archive' => false,
73          'supports' => array( 'title', 'editor', 'author', 'thumbnail', 'custom-fields',
74  'comments' ),
75          'rewrite' => array( 'slug' => 'ateliers' ),
76          'menu_position' => 5,
77          'menu_icon' => 'dashicons-book',
78          )
        );
79  }
80  [...]
```



Le paramètre 'supports' indique les champs qui devront être présents dans le formulaire et les contenus que ce type de « *posts* » devra recenser.

1. `'title'` : Titre du post
2. `'editor'` : Zone d'édition
3. `'custom-field'` : Champ et valeurs spécifiques
4. `'comments'` : Gestion des commentaires
5. `'author'` : Choix de l'auteur
6. `'thumbnail'` : Image associée

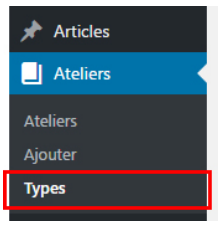
Il est également possible de lui indiquer une position par le paramètre `'menu_position'` ou lui attribuer une icône à l'aide du paramètre `'menu_icon'`.



En savoir plus : https://codex.wordpress.org/Function_Reference/register_post_type

Ces nouveaux « *posts* » peuvent être classifiés et organisés par catégories comme le sont les articles. Pour cela, la fonction `register_taxonomy()` permet de générer cette catégorisation spécifique. Le second paramètre transmis dans cette fonction sert à indiquer le type de post auquel cette catégorisation s'associe.

class.myplugin.php




```
62  [...]
63  private static function create_posts_type()
64  {
80      [...]
81      register_taxonomy(
82          'genre',
83          'ateliers',
84          array(
85              'hierarchical' => true,
86              'label' => 'Types',
87              'query_var' => true,
88              'rewrite' => array( 'slug' => 'types', 'with_front' => false )
89          )
90      );
91  }
92  [...]
```

Pour son activation, il suffit de faire appel à la méthode qui dispose des fonctions `register_post_type()` et `register_taxonomy()`. Dans l'exemple la méthode `create_post_type()` sert à cette fin.

class.myplugin.php

```
6   [...]
7   public static function init()
8   {
9       [...]
13              self::create_posts_type();  // Post Type
14        [...]
15      }
16  [...]
```

Interfaces et menus de l'administration

Hormis la solution par la fonction `register_post_type()` qui permet de créer des type de posts sur mesure, l'interaction avec l'administration de Wordpress implique de générer les rubriques d'accès aux composantes de l'extension et de concevoir ses interfaces. Pour aider à la gestion de cette organisation des contenus la classe `Myplugin_admin()` a été prévue dans le modèle précédemment proposé pour l'administration de l'extension dans le fichier class.myplugin.admin.php.

Cette classe dispose d'un constructeur qui déclenche les opérations au chargement de la classe et deux méthodes, une première pour l'affichage du menu et la gestion des interfaces comme précédemment décrit.

class.myplugin.admin.php

```
1    <?php
2
3    class Myplugin_Admin{
4
5        private $datas;
6        private $action;
7        private $view;
8        private $viewContent;
9
10       public function __construct()
11       {
12       }
13
14       private function my_plugin_admin_menu()
15       {
16       }
17
18       public function my_plugin_admin_display()
19       {
20       }
21   }
```

Des attributs prévus pour le transfert de données destinées à l'interface (view) avec `$datas`, l'archivage de l'action courante avec `$action`, le choix de la vue avec `$view` et le contenu de la vue avec `$viewContent`. Ces attributs seront utiles pour assurer une gestion modulaire des interfaces (vues) et de la séquence des opérations à mener.

Ajouter une rubrique dans le menu de l'administration du CMS

Wordpress propose une série de fonctions qui permettent d'insérer et organiser les rubriques et sous rubriques dans le menu principal du CMS. Les fonctions proposées en l'emplacement de la rubrique dans le menu de l'administration : `add_dashboard_page()`, `add_posts_page()`, `add_media_page()`, `add_pages_page()`, `add_comments_page()`, `add_theme_page()`, `add_plugins_page()`, `add_users_page()`, `add_management_page()`, `add_options_page()`.

La création d'un menu peut également s'effectuer avec la fonction `add_menu_page()`, les paramètres suivants peuvent sont à indiquer dans cet ordre :

- *(obligatoire)* (chaîne de caractères) le titre de la page.
- *(obligatoire)* (chaîne de caractères) le libellé du menu.
- *(obligatoire)* (chaîne de caractères) l'intitulé des droits que doit disposer l'utilisateur pour accéder au menu. Pour découvrir la liste : https://codex.wordpress.org/Roles_and_Capabilities
- *(obligatoire)* (chaîne de caractères) emplacement du fichier appelé ou un identifiant unique dans le cas de l'appel d'une fonction au paramètre suivant. Celui-ci peut être le nom de l'extension (en minuscule et sans espace).

  Exemple : Emplacement du fichier appelé

  ```
  add_menu_page( 'Extension', 'Ext', 'manage_options', MY_PLUGIN_DIR . 'ext.php', null );
  ```

  Exemple : Identifiant unique

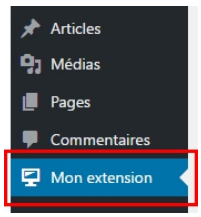
  ```
  add_menu_page( 'Extension', 'Ext', 'manage_options', 'extension', 'ext_method' );
  ```

- *(obligatoire)* (chaîne de caractères) la fonction à appeler lors du rendu de la page d'accès depuis le menu du CMS. Celle-ci est *`null`* dans le cas où le précédent paramètre fait référence à un fichier appelé.
- (chaîne de caractères) l'icône à afficher dans le menu. Peut-être le chemin vers un fichier Wordpress dispose également d'une librairie d'icônes, elle s'identifie par un nom identifiant. Pour découvrir la librairie : https://developer.wordpress.org/resource/dashicons
- (entier) la position dans le menu.

Dans le cas de l'appel d'une fonction au sein d'une classe, celle-ci est référée par `array( $this, 'my_plugin_admin_display' )`. En occurrence, c'est la méthode `my_plugin_admin_menu()` servant à la gestion des interfaces qui est appelée.

class.myplugin.admin.php

```
..    [...]
10        public function __construct()
11        {
12          add_action('admin_menu', function(){$this->my_plugin_admin_menu();});
13        }
14
15        private function my_plugin_admin_menu()
16        {
17            add_menu_page( 'Mon extension', 'Mon extension', 'manage_options',
      'my_plugin', array( $this, 'my_plugin_admin_display' ), 'dashicons-
      desktop', 26 );
18        }
..    [...]
```



Les URL et la navigation entre les interfaces d'une extension

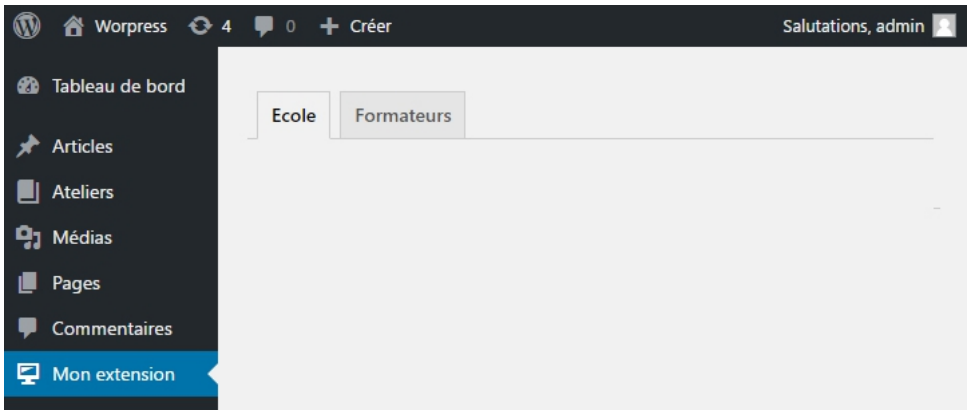
L'administration de Wordpress utilise la variable `page` qui par la méthode de transfert *GET* permet de faire référence à une extension. Dans l'exemple de l'extension présenté, `page` a comme valeur `my_plugin` puisqu'il s'agit du nom donné à cette extension.

Les autres variables transmises en par la méthode *GET* peuvent être récupérées par l'extension en PHP par l'entremise des variables superglobales générées. Dans l'exemple qui suit deux onglets donnes accès à des interfaces différentes. Pour cela, une variable `tab` a été ajoutée à l'URL.

Ce qui donne l'URL `?page=my_plugin&tab=ecole`

views/admin.php

```
1   <?php
2   if ( !current_user_can( 'manage_options' ) )
3   {
4       wp_die( __( 'Ce fichier ne peut pas être accédé directement.' ) );
5   }
6   ?>
7   <div class="wrap">
8
9       <h2 class="nav-tab-wrapper">
10          <a href="?page=my_plugin&tab=ecole" class="nav-tab <?php echo $this->view ==
    'ecole' ? 'nav-tab-active' : ''; ?>">Ecole</a>
11          <a href="?page=my_plugin&tab=formateurs" class="nav-tab <?php echo ( $this->view ==
    'formateurs' || $this->view == 'form-formateur' ) ? 'nav-tab-active' : ''; ?>">Formateurs</a>
12      </h2>
13
14      <?php include $this->viewContent; ?>
15
16  </div>
```



La gestion des interfaces

Le fichier admin.php dispose du contenant HTML qui sera appelé par chaque interface de l'extension. L'attribut `$this->viewContent` disposera du contenu des autres vues qui devront s'afficher dans ce contenant.

Le méthode `my_plugin_admin_display()` détermine l'interface a afficher en récupérant la valeur transmise par la variable tab et transmettre à l'attribut `$this->view` la valeur de l'interface correspondante. Comme indiqué dans l'exemple ci-dessous.

class.myplugin.admin.php

```
..    [...]
19        public function my_plugin_admin_display()
20        {
21            $this->view     = ( isset( $_GET[ 'tab' ] ) ) ? $_GET[ 'tab' ] : 'ecole';
22
23            $this->action   = ( isset( $_GET[ 'action' ] ) ) ? $_GET[ 'action' ] : '';
24
25            switch( $this->view )
26            {
27                case 'ecole':
28                   $this->viewContent = MY_PLUGIN_PATH . 'views/form-ecole.php';
29                break;
30
31                case 'form-formateur':
32                    $this->my_plugin_coach_insert();
33                    if( isset( $_GET['post'] ) )
34                    {
35                        $this->my_plugin_coachs( $_GET['post'] );
36                    }
37                    $this->viewContent = MY_PLUGIN_PATH . 'views/form-formateur.php';
38                break;
39
40                case 'formateurs':
41                    $this->my_plugin_coach_delete();
42                    $this->my_plugin_coachs();
43                    $this->viewContent = MY_PLUGIN_PATH . 'views/list-formateurs.php';
44                break;
45
46                default:
47                   $this->viewContent = MY_PLUGIN_PATH . 'views/form-ecole.php';
48                break;
48            }
50
51            include MY_PLUGIN_PATH . 'views/admin.php';
52        }
..      [...]
```

Les données singulières (options)

Dans le cas où une faible quantité de données sont à conserver, l'utilisation de données singulières peuvent s'avérer utile. Ces données se logent dans la table options et ne disposent que d'une seule valeur. Elles sont pratiques pour définir des paramètres pour l'administration de l'extension. Elles sont simples à générés grâce à la fonction `register_setting()` et tout aussi simple à récupérer avec la fonction `get_option()`.

Dans l'exemple développé précédemment, la déclaration des options par la fonction `register_setting()` peut être déclarée dans la classe principale au chargement de l'extension. En occurrence, la méthode `my_plugin_settings()` est prévue à cette fin dans cette exemple. L'appel de cette méthode peut se faire depuis la méthode statique init() prévue pour la gestion des opération de démarrage de l'extension.

class.myplugin.php

```
..  [...]
7       public static function init()
8       {
9           if ( ! self::$initiated )
10          {
11              self::init_hooks();
12
13              add_action( 'admin_init', function(){ self::my_plugin_settings(); } );
14          }
15      }
..  [...]
59      private static function my_plugin_settings()
60      {
61          register_setting( 'my-plugin-settings-group', 'name' );
62
63          register_setting( 'my-plugin-settings-group', 'phone' );
64
65          register_setting( 'my-plugin-settings-group', 'email' );
66      }
..  [...]
```

Il est utile de grouper les options en leur indiquant le même premier paramètre. Dans l'exemple qui précède il s'agit du groupe `'my-plugin-settings-group'`. Ceci facilite et protège la récupération des données.

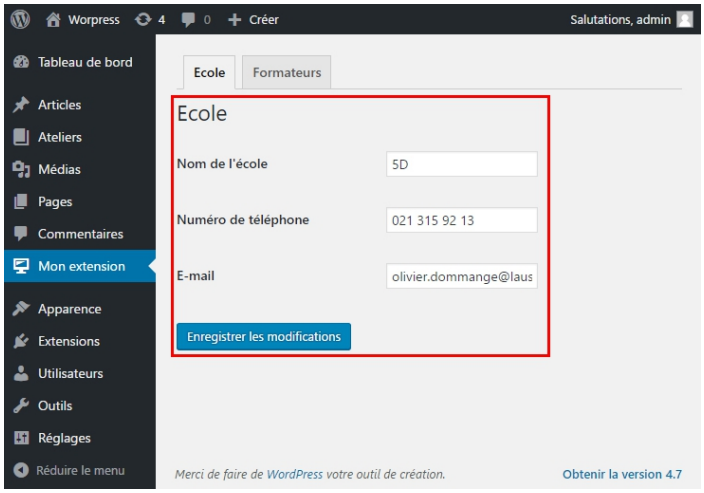
L'interface pour l'édition contiendra un champ pour chaque option et devra disposer de la fonction `settings_fields()`. Celle-ci doit être contenue dans la balise HTML `<form>`. Elle récupère les données provenant d'un groupe pour les disposer dans le formulaire.

Ce formulaire transmet les données au fichier options.php comme indiqué dans l'attribut `action` de la balise <form>. Comme ceci : `<form method="post" action="options.php">`.

views/form-ecole.php

```php
<?php
if ( !current_user_can( 'manage_options' ) )
{
    wp_die( __( 'Ce fichier ne peut pas être accédé directement. ) );
}
?>
<h1>Ecole</h1>
<form method="post" action="options.php">
    <?php settings_fields( 'my-plugin-settings-group' ); ?>
    <table class="form-table">
        <tr valign="top">
        <th scope="row">Nom de l'école</th>
        <td><input type="text" name="name" value="<?php echo esc_attr( get_option('name') );
?>" /></td>
        </tr>

        <tr valign="top">
        <th scope="row">Numéro de téléphone</th>
        <td><input type="text" name="phone" value="<?php echo
esc_attr( get_option('phone') ); ?>" /></td>
        </tr>

        <tr valign="top">
        <th scope="row">E-mail</th>
        <td><input type="text" name="email" value="<?php echo
esc_attr( get_option('email') ); ?>" /></td>
        </tr>
    </table>

    <?php submit_button(); ?>

</form>
```

La fonction `esc_attr()` sécurise les données à afficher le formulaire. La fonction `submit_button()` génère le bouton d'enregistrement des données du formulaire.



Interactions avec la base de données

Wordpress permet une interaction par l'entremise de la variable globale `$wpdp` instanciée de la classe du même nom. Voici une description des principales méthodes de cette classe.

`get_var()` Récupère une donnée à partir d'une requête de sélection.

```
$name = $wpdb->get_var( "SELECT link_name FROM $wpdb->links WHERE id = 10" );
```

`prepare()` Prépare les données pour une requête en les assainissant et supprimant les risques d'injection SQL. Les types de données doivent être indiqués : `'%s'` : String, `'%d'` : Interger, `'%f'` : Float.

```
$wpdb->prepare('INSERT INTO '.$wpdb->prefix.'table VALUES(null, \'John\')');
```

`query()` Transmet une requête à la base de données. Se combine avec la méthode prepare() pour prévenir des injections SQL.

```
$wpdb->query( $preparedquery );
```

`get_row()` Récupère une rangée de données et la transmet sous forme d'objet (peut être défini sous la forme de *array*).

```
$values = $wpdb->get_row( "SELECT * FROM $wpdb->links WHERE link_id = 10" );
```

`get_results()` Récupère un groupe de données et les transmet sous forme de *array* composés d'objets (peut être défini sous la forme de *array*).

```
$values = $wpdb->get_results( "SELECT * FROM $wpdb->posts" );
```

`insert()` Effectue l'insertion de données. Est généré l'attribut `$wpdb->insert_id` contenant la rangée des données de la dernière insertion.

```
$wpdb->insert( $wpdb->prefix . 'table',
    array('column1' => 'value1', 'column2' => 123 ),
    array('%s', '%d')
);
```

`update()` Effectue la mise à jour de données. Le second *array* passé comme paramètre indique les conditions d'identification de la rangée dans laquelle les données sont à mettre à jour.

```
$wpdb->update( $wpdb->prefix . 'table',
    array('column1' => 'value1', 'column2' => 123 ),
    array('id' => 12 ),
    array('%s', '%d') ,
    array('%d')
);
```

`replace()` Effectue le remplacement de données. La première valeur étant l'identifiant de référence utile à l'identification de la rangée dans laquelle les données sont à remplacer.

```
$wpdb->replace( $wpdb->prefix . 'table',
    array('id' => 12, 'column1' => 'value1', 'column2' => 123 ),
    array('%d', '%s', '%d')
);
```

`delete()` Supprime des données selon la condition d'identification indiquée dans le premier *array*.

```
$wpdb->delete( $wpdb->prefix . 'table',
    array('id' => 12, ),
    array('%d')
);
```

En savoir plus : https://codex.wordpress.org/Class_Reference/wpdb

L'utilisation des méthodes de la classe `wpdb()` peut s'employer comme suit en référence à l'exemple précédemment présenté.

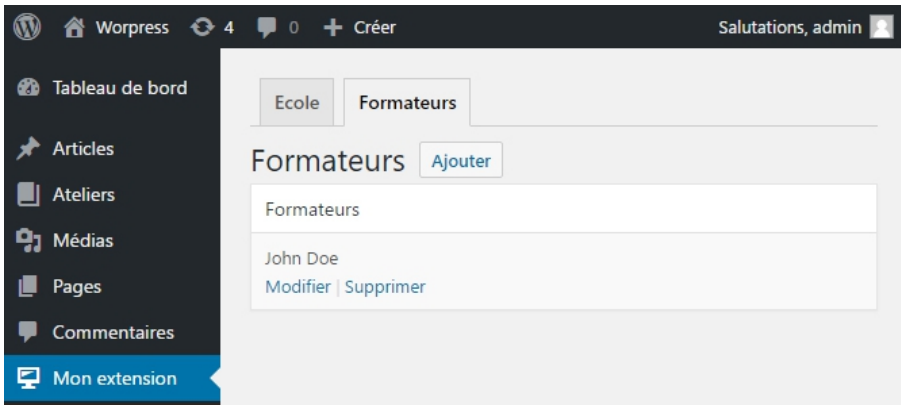
class.myplugin.admin.php

```
..    [...]
56    private function my_plugin_coachs( $id = null )
57    {
58        global $wpdb;
59
60        $where = ( isset( $id ) ) ? ' WHERE id_coach = \'' . $id . '\'' : '';
61
62        $this->datas = $wpdb->get_results('SELECT * FROM ' . $wpdb->prefix . 'coachs' .$where);
63    }
64
65    private function my_plugin_coach_insert()
66    {
67        if( isset( $_POST[ 'name_coach' ] ) && !empty( $_POST[ 'name_coach' ] ) )
68        {
69            global $wpdb;
70
71            if( isset( $this->action ) && $this->action === 'add' )
72            {
73                $wpdb->insert( $wpdb->prefix . 'coachs',
74                        array( 'name_coach' => $_POST[ 'name_coach' ]),
75                        array( '%s' )
76                    );
77                $this->my_plugin_coachs( $wpdb->insert_id );
78                $this->action = 'edit';
79            }
80            else if( isset( $this->action ) && $this->action === 'edit' )
81            {
82                $wpdb->update( $wpdb->prefix . 'coachs',
83                        array( 'name_coach' => $_POST[ 'name_coach' ]),
84                        array( 'id_coach' => $_POST[ 'id_coach' ] ),
85                        array( '%s' ),
86                        array( '%d' )
87                    );
88                $this->my_plugin_coachs( $_POST['id_coach'] );
89            }
90        }
91    }
92
93    private function my_plugin_coach_delete()
94    {
95        if( isset( $this->action ) && $this->action === 'delete' )
96        {
97          global $wpdb;
98
100         $wpdb->delete($wpdb->prefix.'coachs', array('id_coach'=>$_GET['post']), array('%d'));
101       }
102   }
..    [...]
```

Les interfaces qui dispose ensuite des données transmises depuis la base de données. Les interfaces utilisent les balises HTML et les classes CSS de référence selon Wordpress.

views/list-formateurs.php

```php
<?php
if ( !current_user_can( 'manage_options' ) )
{
    wp_die( __( 'Ce fichier ne peut pas être accédé directement.' ) );
}
?>
<h1>Formateurs <a href="?page=my_plugin&tab=form-formateur&action=add" class="page-title-action">Ajouter</a></h1>

<table class="widefat fixed" cellspacing="0">
<thead>
<tr>
    <th id="columnname" class="manage-column column-columnname" scope="col">Formateurs</th>
</tr>
</thead>

<tbody>
<?php foreach( $this->datas as $n => $data ) { ?>

    <tr class="<?php echo ( $n % 2 === 0 ) ? 'alternate' : ''; ?>">
        <td class="column-columnname">
            <?php echo $data->NameCoach; ?>
            <div class="row-actions">
                <span><a href="?page=my_plugin&tab=form-formateur&action=edit&post=<?php echo $data->id_coach; ?>">Modifier</a> |</span>
                <span><a href="?page=my_plugin&tab=formateurs&action=delete&post=<?php echo $data->id_Coach; ?>">Supprimer</a></span>
            </div>
        </td>
    </tr>

<?php } ?>
</tbody>
</table>
```

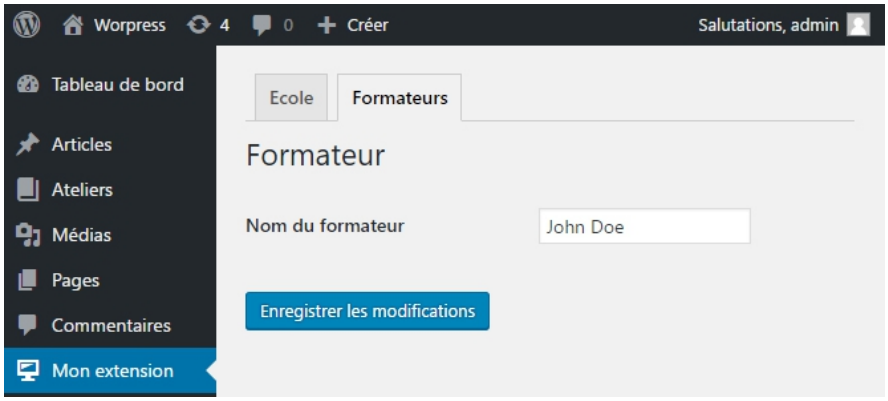


views/form-formateurs.php

```php
<?php
if ( !current_user_can( 'manage_options' ) )
{
    wp_die( __( 'Ce fichier ne peut pas être accédé directement.' ) );
}
?>
<h1>Formateur</h1>
<form method="post" action="?page=my_plugin&tab=form-formateur&action=<?php echo $this->action; ?>">

    <input type="hidden" name="id_coach" value="<?php echo ( isset( $this->datas[0] ) ) ? esc_attr( $this->datas[0]->id_coach ) : ''; ?>" />

    <table class="form-table">
        <tr valign="top">
        <th scope="row">Nom du formateur</th>
        <td><input type="text" name="name_coach" value="<?php echo ( isset( $this->datas[0] ) ) ? esc_attr( $this->datas[0]->name_coach ) : ''; ?>" /></td>
        </tr>
    </table>

    <?php submit_button(); ?>

</form>
```

Ce formulaire transmet par la méthode GET les paramètres `page`, `tab` et `action` par l'entremise de l'attribut `action` de la balise `<form>`. Ceci indique le traitement à effectuer des données et spécifie s'il s'agit d'une insertion ou d'un mise à jour de données. Comme ceci : `<form method="post" action="?page=my_plugin&tab=form-formateur&action=add">`.

La fonction `esc_attr()` sécurise les données à afficher le formulaire. La fonction `submit_button()` génère le bouton d'enregistrement des données du formulaire.



Créer un *shortcode*

Le *shortcode* permet d'introduire dans un post un élément étranger à ce post. Un exemple d'application serait d'introduire une galerie d'images provenant d'une extension servant à les composer en insérant un *shortcode* rappelant l'extension et faisant référence à une des galeries existantes. Ce qui donnerait quelque chose comme `[galerie name="logos"]`. Un *shortcode* peut contenir plusieurs paramètres. Ils sont définis selon les indications transmises lors de la composition du *shortcode*.

La création d'un *shortcode* se fait par l'appel de la fonction `add_shortcode()`. Celle-ci définit le nom du *shortcode* en plus de transmettre les données générées lors de l'appel du shortcode par l'entremise d'une fonction. Voici un exemple simple qui génère un *shortcode* du nom de `[foobar]`.

```
function foobar_func()
{
        return "foo and bar";
}
add_shortcode( 'foobar', 'foobar_func' );
```

Le même exemple mais avec l'utilisation d'une fonction anonyme.

```
add_shortcode( 'foobar', function(){ return "foo and bar"; } );
```

Voici un nouvel exemple générant cette fois des attributs au moyen de la fonction `shortcode_atts()`. Le résultat sera visible lorsque le *shortcode* `[foobar foo="ma valeur"]`.

```
function bartag_func( $atts )
{
    $a = shortcode_atts( array(
            'foo' => 'foo default value',
            'bar' => 'bar default value' ]
            ), $atts );

    return "foo = {$a['foo']} | bar = {$a['bar']}";
}
add_shortcode( 'bartag', 'bartag_func' );
```

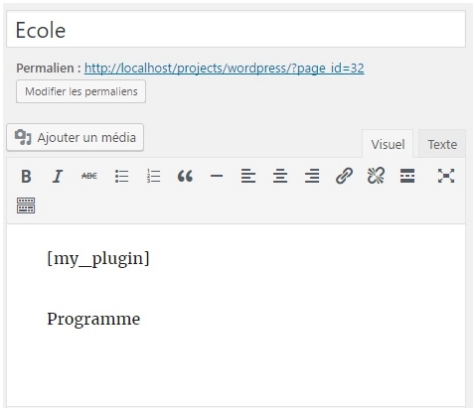
Dans l'exemple précédemment développé, le *shortcode* sera généré dans la classe `Myplugin_Front()` qui permet d'insérer des contenus dans la page du site.

class.myplugin.front.php

```
1   <?php
2
3   class Myplugin_Front{
4
5       private $datas;
6
```

```
7       public function __construct()
8       {
9           add_action( 'wp_enqueue_scripts', function(){ $this->my_enqueued_assets(); } );
10
11          add_shortcode( 'my_plugin', array( $this, 'ecole_content' ) );
12      }
13
14      public function ecole_content( $atts )
15      {
16          $this->datas = shortcode_atts( array(
17              'ecole'     => esc_attr( get_option('name') ),
18              'tel'       => esc_attr( get_option('phone') ),
19              'email'     => esc_attr( get_option('email') )
20              ), $atts );
21
22          include MY_PLUGIN_PATH . 'views/front-ecole.php';
23      }
24  }
```

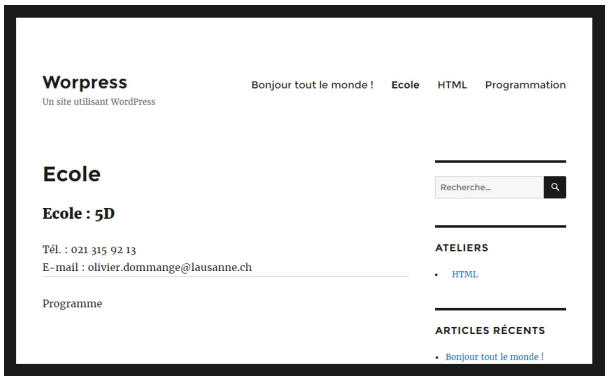
Le code précédent permet d'insérer le shortcode [my_plugin] dans un post et d'afficher les options générés dans un précédent exemple.



L'affichage se définit dans le fichier front-end.php comme indiqué dans la méthode `ecole_content()`.

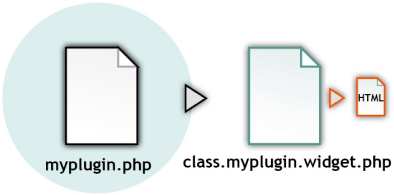
views/front-ecole.php

```
1   <?php
2   if ( !current_user_can( 'manage_options' ) )
3   {
4       wp_die( __( 'Ce fichier ne peut pas être
    accédé directement.' ) );
5   }
6   ?>
7   <h2>Ecole : <?php echo $this->datas['ecole'];
    ?></h2>
8   Tél. : <?php echo $this->datas['tel']; ?>
9   <br>
10  E-mail : <?php echo $this->datas['email']; ?>
11  <hr>
```



Créer un Widget

La création d'un Widget a la particularité qu'il faille surcharger la classe `WP_Widget()`. Selon l'exemple précédemment présenté, il est question de créer la classe `My_Plugin_Widget()`. Elle sera référée depuis le fichier principal de l'extension myplugin.php.



myplugin.php

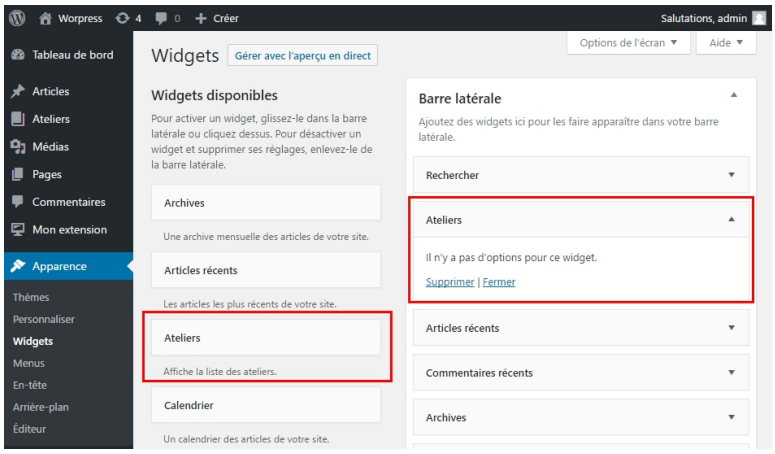
```
..  [...]
27  add_action( 'widgets_init', function(){ register_widget( 'My_Plugin_Widget' ); } );
```

La classe My_Plugin_Widget() est étend la classe WP_Widget() afin de surcharger cette dernière. La déclaration du Widget se fait depuis le constructeur parent à qui sont transmis le nom et la description du widget.

class.myplugin.widget.php

```
1   <?php
2
3   class My_Plugin_Widget extends WP_Widget
4   {
5
6       public function __construct()
7       {
8           parent::__construct( 'mon_extension', 'Ateliers', array(
9               'description' => 'Affiche la liste des ateliers.'
10              ) );
11      }
12  }
```

Ceci suffit pour que le Widget soit reconnu par l'administration du CMS et soit utilisé.



Pour générer l'interface prévue pour le site par le Widget, il suffit de surcharger la méthode `widget()`. Des paramètres sont disponibles en fonction des éventuelles options attribuées au Widget lors de la définition du constructeur parent. Pour en savoir plus sur ces options : https://developer.wordpress.org/reference/classes/wp_widget/__construct/

class.myplugin.widget.php

```
..  [...]
13      public function widget( $args, $instance )
14      {
15          $myquery = new WP_Query( array( 'post_type'=>'ateliers' ) );
16
17          include MY_PLUGIN_PATH . 'views/widget-ateliers.php';
18      }
..  [...]
```

Dans cet exemple, sont récupérées les « *posts* » du type ateliers créés dans un exemple précédent. Puis, introduit, le fichier widget-ateliers.php qui contient les contenus HTML à afficher.

views/front-ecole.php

```
1   <?php
2   if ( !current_user_can( 'manage_options' ) )
3   {
4       wp_die( __( 'Ce fichier ne peut pas être accédé directement.' ) );
5   }
6   echo $args['before_widget'];
7   echo $args['before_title'];
8   echo $args['widget_name'];
9   echo $args['after_title'];
10
11  if( $myquery->have_posts() ):
12
13      while( $myquery->have_posts() ): $myquery->the_post();
14
15          echo '<li><a href="'; the_permalink(); echo '">'; the_title(); echo '</a></li>';
16
17      endwhile;
18
19  else:
20
21      echo '<p>Aucun atelier</p>';
22
23  endif;
24
25  echo $args['after_widget'];
```

Les variables `$args['before_widget']`, `$args['before_title']`, `$args['widget_name']`, `$args['after_title']` et `$args['after_widget']` disposent des informations relatives à la génération du contenant du Widget.

Lier des fichiers CSS et Javascript au site

L'association de fichiers CSS et Javascript peut-être associé au site dès le moment où l'extension est activée. Par l'utilisation des fonctions `wp_enqueue_style()` pour les feuilles de styles et `wp_enqueue_script()` pour les fichiers Javascript.

Celles-ci peuvent être introduites par leur insertion dans une méthode prévue pour leur déclenchement, `wp_enqueued_assets()` dans l'exemple suivant. Cette méthode pourra être appelée par le constructeur de la classe `Myplugin_Front()` dans laquelle elle est contenue.

class.myplugin.front.php

```
..  [...]
7       public function __construct()
8       {
9           add_action( 'wp_enqueue_scripts', function(){ $this->my_enqueued_assets(); } );
10
11          add_shortcode( 'my_plugin', array( $this, 'ecole_content' ) );
12      }
..  [...]
25      private function my_enqueued_assets()
26      {
27          wp_enqueue_style( 'fonts', '//fonts.googleapis.com/css?family=Ubuntu' );
28
29          wp_enqueue_script( 'scripts', MY_PLUGIN_URL . 'assets/scripts.js', '', '', true );
30      }
31  }
```

La migration du CMS

La migration d'un site implique de disposer de différentes informations au sujet du futur serveur (chez l'hébergeur). Une première information sont les codes d'accès FTP pour le transfert des fichiers. La seconde concerne la connexion à la base de données qui doit normalement être créée via l'administration de l'hébergement tout comme le nom d'utilisateur et le mot de passe pour y accéder.

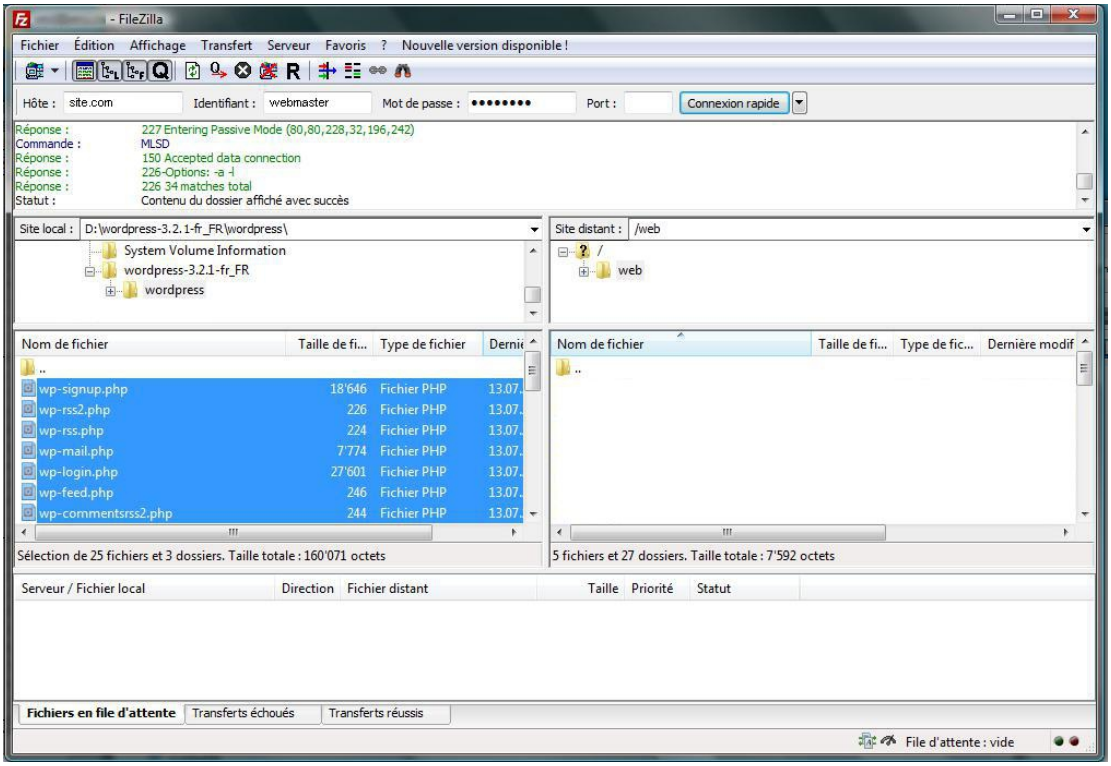
1. **Indiquer les accès à la base du futur serveur dans le fichier wp-config.php.**

   Ce fichier contient les informations de connexion à la base de données qui devront être paramétrées.

```
define('DB_NAME', 'monsite');          // Le nom de la base de données
define('DB_USER', 'root');             // Nom d'utilisateur
define('DB_PASSWORD', 'mysql');        // Mot de passe
define('DB_HOST', 'localhost');        // Nom de l'hôte
```

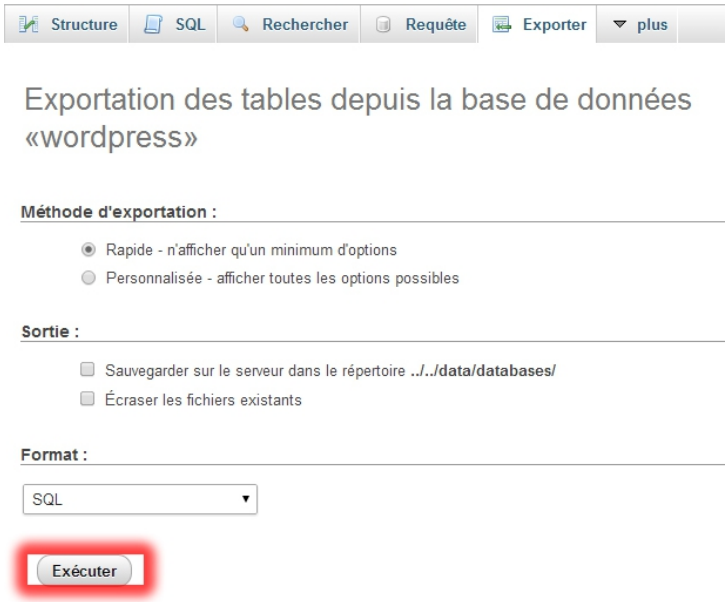
2. **Transférer les fichiers du site via FTP**

   Cette opération s'effectue à l'aide d'un outil (client) FTP (Filezilla sur PC ou Cyberduck sur Mac). Les fichiers en local sont transférés sur le serveur.



3. Transférer la base de données

Le transfert consiste à exporter la structure des tables ainsi que les données au format sql. Un outil d'exportation est disponible dans la console d'administration des bases de données PhpMyAdmin. En indiquant la base de données concernée par l'exportation, il suffit de cliquer sur l'onglet [Export]. Aucune spécification n'est nécessaire. Seule l'exécution de l'exportation peut être effectuée en cliquant sur le bouton [Exécution].

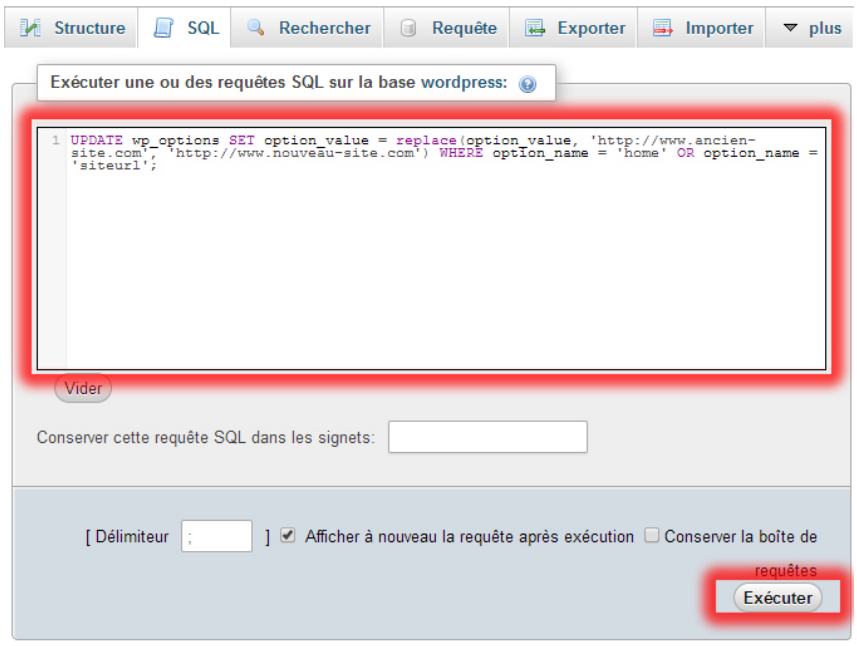


Une fois le fichier sql créé, il est possible d'importer les données dans la base de données du futur site.

L'importation s'effectue également dans la console d'administration des bases de données PhpMyAdmin. L'onglet [Import] met à disposition un outil qui permet de sélectionner et exécuter le fichier d'importation au format sql.

4. Corriger les liens du site

WordPress conserve l'adresse des liens du site dans la base de données. En migrant le site ceux-ci doivent être corrigés... Pour cela il faudra utiliser la console d'administration de la base de données PhpMyAdmin afin d'exécuter des requêtes qui effectueront les corrections nécessaires. L'onglet [sql] permet d'exécuter des requêtes depuis la console.



Les requêtes :

Dans ces requêtes doit être remplacé `http://www.ancien-site.com` par l'URL de l'ancien site et `http://www.nouveau-site.com` par l'URL du nouveau site

Changer l'URL du site

```
UPDATE wp_options SET option_value = REPLACE(option_value, 'http://www.ancien-site.com', 'http://www.nouveau-site.com') WHERE option_name = 'home' OR option_name = 'siteurl';
```

Changer le GUID (Global Unique Identifier)

```
UPDATE wp_posts SET guid = REPLACE (guid, 'http://www.ancien-site.com', 'http://www.nouveau-site.com');
```

Changer l'URL des liens, des images et des documents

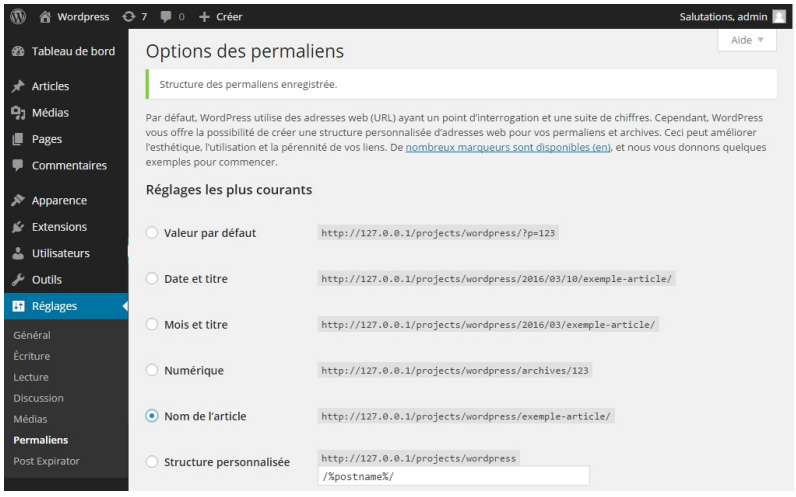
```
UPDATE wp_posts SET post_content = REPLACE (post_content, 'http://www.ancien-site.com', 'http://www.nouveau-site.com');
```

Changer l'URL méta-données

```
UPDATE wp_postmeta SET meta_value = REPLACE (meta_value, 'http://www.ancien-site.com','http://www.nouveau-site.com');
```

5. **Corriger les informations du fichier .htaccess**

WordPress transforme l'adresse des liens du site en fonction de l'option définie dans le réglage des permaliens [Réglages/Permaliens].

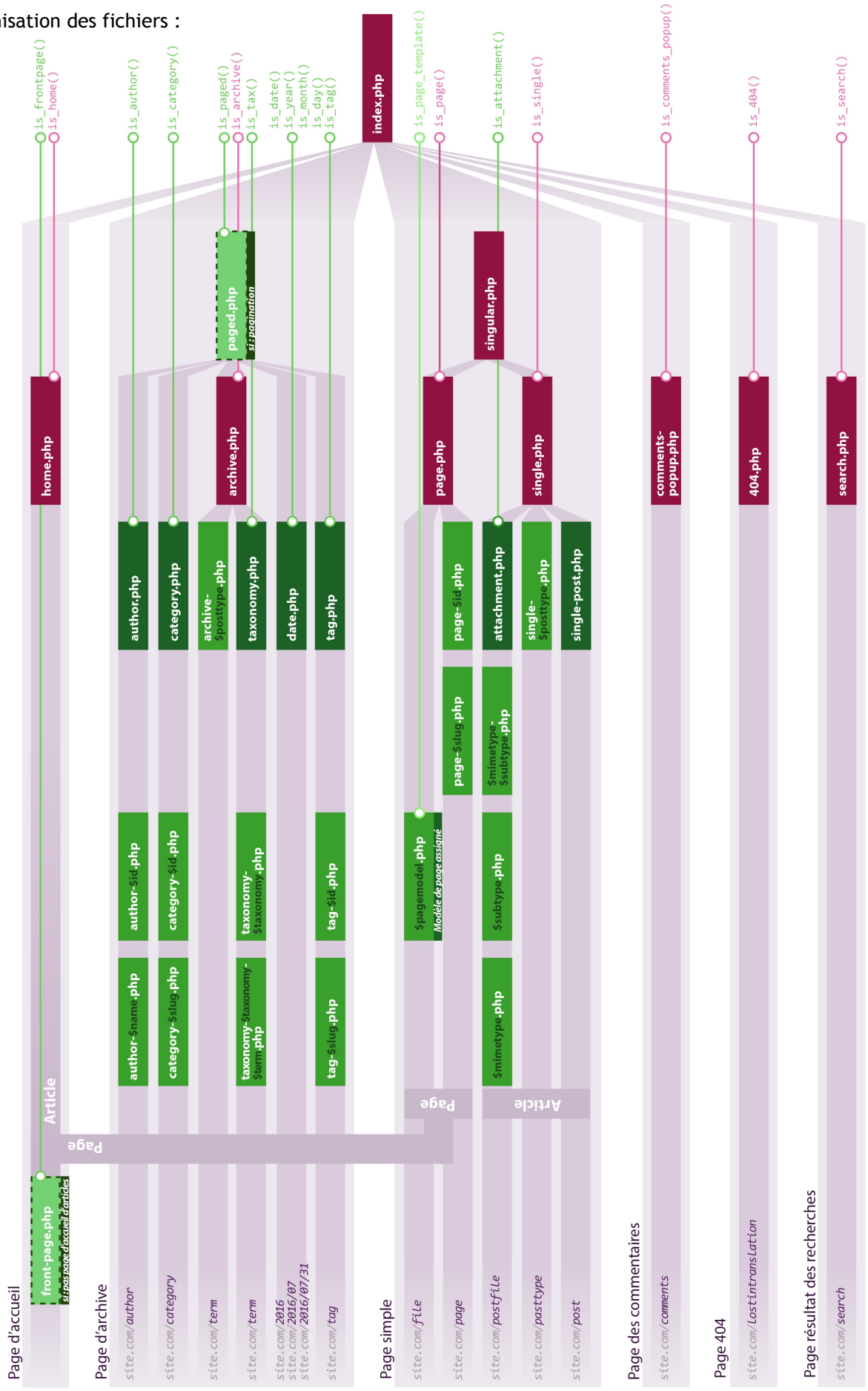


Les options choisies, autres que la « Valeur par défaut » génère un fichier .htaccess à la racine du site qui transmet les paramètres de l'URL au CMS sous sa nouvelle forme. Lors de la migration ces informations doivent être modifiées pour les faire correspondre à la structure des répertoires du nouveau serveur.

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteBase /chemin/serveur/
RewriteRule ^index\.php$ - [L]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /chemin/serveur/index.php [L]
</IfModule>
```

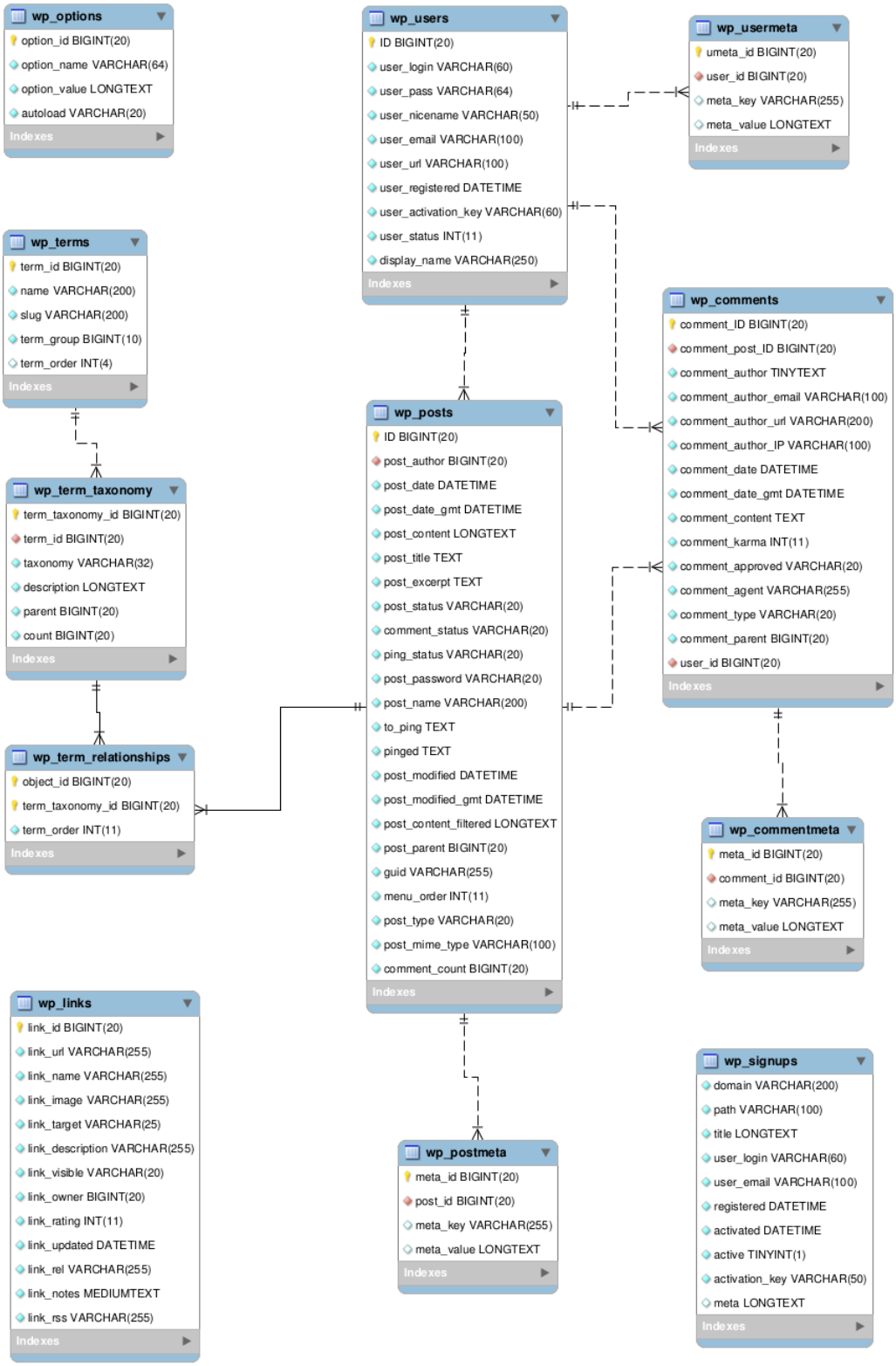
La hiérarchie des thèmes

Organisation des fichiers :



La structure de la base de données

Voici la structure de la base de données du CMS Wordpress :



Source : Wordpress.org (https://codex.wordpress.org/File:WP3.9.4-ERD.png)