12 Construire une extension évoluée

Objectif de l'extension

Dans ce chapitre, nous allons créer une extension "évoluée", autrement dit une extension ajoutant plusieurs fonctionnalités et faisant appel à différentes API de WordPress.

Ici, nous allons mettre au point une extension permettant de gérer une liste de petites annonces. Cette extension, nommée Simple Classifieds (en anglais, "petite annonce" s'écrit *classified ads*), gère différentes fonctionnalités que nous détaillerons dans la section suivante.

C'est typiquement le genre d'extension qui pourra être utilisée dans le cadre d'un service en ligne gratuit, par exemple. Une orientation qui tend plus vers le CMS que l'outil de blog…

Quelles fonctionnalités ?

Avant d'écrire ce code, il est impératif que vous vous posiez quelques questions. Ces questions vont permettre de regrouper les fonctionnalités de votre extension dans différents groupes, selon le type de fonctions sur lesquelles vous travaillerez, mais elles doivent également rappeler les différentes pages de l'extension.

Voici une liste non exhaustive des questions qui peuvent être importantes :

- Quelles sont les fonctionnalités ?
- Quelles sont les pages indispensables de l'interface d'administration ?
- Quels utilisateurs de WP vont pouvoir modifier les réglages de votre extension ?
- Quels utilisateurs de WP vont pouvoir utiliser l'extension ?
- Comment s'affichera le contenu de l'extension dans le thème ?
- Quelle stratégie vis-à-vis du référencement ?
- Où stocker les données ?
- Quelles fonctionnalités seront disponibles depuis la page de rédaction ? Etc.

Cette liste est un simple exemple. Avec l'expérience que vous allez acquérir, vous ajouterez probablement de nouvelles questions au fur et à mesure de vos développements.

Revenons à notre exemple, Simple Classifieds, et répondons aux questions suivantes.

Quelles sont les fonctionnalités ?

L'extension doit pouvoir gérer l'ajout, l'édition et la suppression des annonces. L'extension contiendra également un widget pour afficher les dernières annonces et un shortcode permettant d'implémenter un formulaire d'ajout public dans la page de notre choix.

Quelles sont les pages indispensables de l'interface d'administration ?

Trois pages sont à prévoir : une page de gestion listant les événements, une page contenant le formulaire d'ajout et d'édition des événements et une page pour les réglages de l'extension. Le listage, l'ajout et l'édition des annonces seront directement gérés par WordPress.

Quels utilisateurs de WP vont pouvoir modifier les réglages de votre extension ?

Seul le rôle administrateur aura la possibilité de modifier les réglages de l'extension.

Quels utilisateurs de WP vont pouvoir utiliser l'extension ?

Les rôles administrateur et éditeur auront la possibilité d'utiliser l'extension, soit ajouter, effacer et modifier les événements.

Comment s'affichera le contenu de l'extension dans le thème ?

Pour afficher les annonces dans le thème de l'utilisateur, vous disposez de différents moyens. Tout d'abord, l'extension se chargera de modifier la Boucle de la page d'accueil pour y afficher les annonces plutôt que les articles de l'installation WordPress. Par ailleurs, vous avez un widget qui affiche les dernières annonces ajoutées.

Quelle stratégie vis-à-vis du référencement ?

L'extension doit être capable de s'adapter à la forme des liens de WordPress. Si les permaliens sont actifs, l'extension doit générer de belles adresses, sinon elle produira des adresses classiques avec des paramètres.

Où stocker les données ?

Les annonces seront stockées dans les tables standard de WordPress, grâce à l'utilisation d'un type de données personnalisé. Sans cela, il nous aurait fallu créer nos propres tables de base de données...

Quelles fonctionnalités seront disponibles depuis la page de rédaction ?

Il y a deux zones possibles de rédaction : la page interne de WordPress, et le formulaire généré par le shortcode de l'extension. Dans les deux cas, l'utilisateur ne pourra saisir qu'un titre et un contenu textuel.

L'exemple que nous utilisons se base sur les types de contenus personnalisés, ce qui simplifie grandement l'extension, tant du point de vue de l'accès à la base que de la mise en place des pages d'administration, qui sont quasi totalement pris en charge par WordPress.
Si vous souhaitez découvrir une extension qui vous explique comment manipuler la base ou mettre en place vos propres pages d'administration, le CD de ce livre contient la première édition du présent chapitre, qui comprend une extension de gestion d'événements, Simple Events. Notez que Simple Events a été écrit pour WordPress 2.7 et ne profite donc pas des nombreuses améliorations des versions suivantes...

Regroupement des fonctionnalités

Une fois que vous avez répondu à ces différentes questions, vous devez regrouper les données par entité logique de WordPress. Une entité logique correspond en fait à un groupement de fonctionnalités selon les bibliothèques de fonctions utilisées ; par exemple, si l'on devait accéder à la base de données, il faudrait passer par l'API WPDB.

Administration

- Ajout des menus dans WordPress.
- Création des nouvelles permissions et attributions aux rôles.
- Création du contenu des trois pages de l'extension.
- Mise en place d'un shortcode permettant de placer un formulaire d'ajout à une page arbitraire.

Accès en base de données

- Totalement pris en charge par WordPress grâce aux types de contenu personnalisés.

Réécriture d'adresse Internet – Rewriting

- Totalement pris en charge par WordPress.

Fonction du thème

- Modification à la volée de la Boucle de la page d'accueil.

Widget

- Création de l'option permettant d'activer ou non le widget.
- Création du widget.

Taxinomie

- Enregistrement de deux taxinomies pour notre type de contenu personnalisé : catégories et mots-clefs.

Architecture de l'extension

Fichier ou dossier ?

Pour architecturer une extension WordPress, deux cas de figure se présentent.

Le premier cas, c'est celui d'une extension simple comme Hello Dolly, que vous pouvez retrouver par défaut dans WordPress. Comme vous avez pu le constater, cette extension ne fait pas grand-chose, de ce fait placer toutes les fonctions dans un seul et même fichier est largement compréhensible.

Le second cas, à l'instar de notre exemple ici, est celui d'une extension évoluée, susceptible de contenir différents fichiers pour la traduction, un widget, une documentation ou encore des modèles de templates. Pour cette raison, il est nécessaire de créer un dossier pour ce type d'extension.

Un gros fichier ? Ou plusieurs petits fichiers ?

Dans le cadre d'une extension assez complexe, il y a deux façons de répartir le code. Soit vous placez tout le code dans un seul fichier PHP, soit vous découpez l'extension par entité logique pour créer un fichier PHP par entité. Les deux techniques ont leurs avantages et leurs défauts.

Par exemple, si vous choisissez de placer toutes les fonctions dans un seul et même fichier, il sera difficile de trouver rapidement une fonction. Une extension un tant soit peu complexe atteindra très rapidement les 1 000, voire 2 000 lignes, pour peu que vous ayez des besoins qui ne soient pas pris en charge par les API de WordPress. D'un autre côté, même avec un grand nombre de fonctions, si ces dernières sont réparties dans différents fichiers thématiques, il sera assez facile de trouver celles qui vous intéressent.

Néanmoins, il n'est pas toujours possible de découper toutes les fonctions dans différents fichiers, pour la simple et bonne raison qu'un développeur peut travailler avec des classes PHP, et ces dernières ne peuvent pas être réparties sur plusieurs fichiers. Dans ce cas précis, la répartition pourra être la suivante : une classe cliente lancée tout le temps, et une classe admin ne démarrant que les fonctions spécifiques à l'interface d'administration seulement lorsque vous vous y trouvez.

Architecture de Simple Classifieds

Ici, pour simplifier le code, l'extension sera principalement développée *via* des fonctions classiques ; seul le widget sera conçu sous la forme d'une classe. De ce fait, l'extension sera architecturée de la façon suivante :

```
simple-classifieds/
  simple-classifieds.php
  readme.txt
  languages/
    simpleclassifiedsfr_FR.mo
    simpleclassifieds-fr_FR.po
    simpleclassifieds.pot
  inc/
    classifieds.init.php
    classifieds.options.php
    classifieds.shortcodes.php
    classifieds.widgets.php
```

Toute l'extension est contenue dans le dossier simple-classifieds. Ce dernier possède deux dossiers et un fichier PHP.

Le fichier PHP simple-classifieds.php contient l'en-tête spécifique aux extensions WordPress. C'est lui qui gère l'inclusion des autres bibliothèques, autrement dit c'est le cœur de l'extension.

Les deux dossiers sont languages et inc. Ils contiennent respectivement les fichiers de traduction de l'extension et les bibliothèques de fonctions nécessaires à l'extension, qui sont au nombre de quatre.

Développement de l'extension

Les bases de l'extension

L'en-tête de WordPress

Comme dans toute extension, il est nécessaire d'avoir l'en-tête des extensions WordPress.

Nous débuterons le fichier simple-classifiedss.php avec ce code :

```
/*
Plugin Name: Simple Classifieds
Plugin URI: http://wordpress.org/extend/plugins/simple-classifieds/
Description: Run your own classifieds website using WordPress.
Version: 1.0
Author: Xavier Borderie
Author URI: http://www.wordpress-fr.net
License: GPL2
Text Domain: simpleclassifieds
*/
```

Cet en-tête peut être suivi de la licence de l'extension et d'une notice de copyright.

Inclusion des différents fichiers

Pour permettre l'inclusion des fonctions contenues dans les différents fichiers, vous devez ajouter les appels PHP des fichiers après l'en-tête de WordPress :

```
require( dirname(__FILE__) . '/inc/classifieds.init.php' );
require( dirname(__FILE__) . '/inc/classifieds.shortcodes.php' );
```

Seuls sont appelés ici les fichiers qui doivent forcément être chargés en même temps que l'extension. D'autres fichiers ne sont chargés qu'en cas de besoin...

Remarquez que nous ne précisons pas le chemin complet des fichiers PHP, nous utilisons à la place la fonction PHP `dirname()` qui retourne le chemin complet du dossier contenant le fichier que nous passons en paramètre. Ici nous passons en paramètre la constante PHP `__FILE__` ; cette constante a comme valeur le chemin complet du fichier où la constante est appelée.

Activation de l'extension

Lors de l'activation de l'extension dans WordPress, nous allons devoir effectuer deux actions : créer le type de contenu et les taxinomies utilisées par nos petites annonces, et initialiser des permissions spécifiques à notre extension.

Les fonctions créant le type de contenu et les taxinomies sont déclenchées par le crochet 'init' :

```
add_action( 'init', 'xb_classifieds_build_post_type' );
add_action( 'init', 'xb_classifieds_build_taxonomies' );
```

Ainsi, ces deux fonctions seront lancées tant que l'extension sera activée.

Pour lancer une fonction PHP uniquement lors de son activation (et donc éviter les multiples appels alors que le code est déjà en place), nous utiliserons la fonction `register_activation_hook()`, qui appellera automatiquement la fonction `xb_classifieds_build_permissions()` lors de l'activation de l'extension.

Ajoutez cette ligne :

```
register_activation_hook( __FILE__, 'xb_classifieds_build_permissions' );
```

Cette fonction doit être placée après l'ajout des fichiers PHP.

Initialisation des permissions

L'extension disposera de deux permissions différentes : une première permettant de gérer les données de l'extension et l'autre permettant de modifier les réglages de l'extension.

Ici, la permission de gestion sera nommée `use_classifieds` et elle sera attribuée aux rôles éditeur et administrateur, tandis que la permission `admin_classifieds` sera attribuée uniquement à l'administrateur, lui permettant de modifier les réglages :

```
// Ajout des permissions
function xb_classifieds_build_permissions() {
  if ( function_exists('get_role') ) {

    // Je récupère l'objet "Rôle administrateur"
    $role = get_role('administrator');

    // Si la permission "use_classifieds" n'existe pas, on l'ajoute.
    if( $role != null && !$role->has_cap('use_classifieds') ) {
      $role->add_cap('use_classifieds');
    }

    // Pareil pour la permission "admin_classifieds"
    if( $role != null && !$role->has_cap('admin_classifieds') ) {
      $role->add_cap('admin_classifieds');
    }

    // On supprime la variable de notre fonction.
    unset($role);

    // On procède de la même façon pour le rôle "Editeur" sauf qu'on lui ajoute
    uniquement la permission "user_classifieds"
    $role = get_role('editor');
    if( $role != null && !$role->has_cap('use_classifieds') ) {
      $role->add_cap('use_classifieds');
    }
```

```
        // On supprime la variable de notre fonction.
        unset($role);
      }
    }
```

Ces permissions vont permettre de sécuriser les actions de l'extension et les menus. WordPress n'affichera que les menus dont l'utilisateur possède la permission. De ce fait, un éditeur ne verra pas le menu Réglages de l'extension.

Initialisation de l'extension

Pour lancer les différentes fonctionnalités de l'extension, il faut initialiser les différents filtres et actions. Pour cela, vous allez regrouper ces appels dans une seule et unique fonction xb_classifieds_`init()` que vous placerez dans le fichier simple-classifieds.php.

C'est la seule fonction que contiendra ce fichier et elle sera placée à la fin du fichier.

```
// Fonction d'initialisation de l'extension
add_action('plugins_loaded', 'xb_classifieds_init');
function xb_classifieds_init() {
  //…
}
```

Vous ajouterez du code à l'intérieur de la fonction au fur et à mesure du développement. Notez que la fonction sera lancée lors de l'action `plugins_loaded`, pas avant !

Mise en place du type de contenu

Nos petites annonces seront stockées non pas au sein d'articles ou de pages, mais dans un nouveau type de contenu créé pour l'occasion et nommé "classifieds" (petite annonce en anglais, voir Figure 12.01).

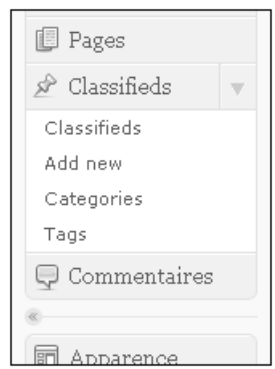

Figure 12.01
Le menu généré par WordPress, avec les taxinomies.

La création d'un type de contenu se fait très rapidement : il suffit d'appeler la fonction `register_post_type()`, et le type de contenu est pris en compte par WordPress, qui

génère automatiquement les menus de l'interface d'administration, ainsi que toute la logique interne nécessaire au bon fonctionnement de ce type de contenu.

Parce que nous sommes attentifs aux détails, nous allons faire en sorte d'intégrer au mieux le type de l'interface. Cela signifie non seulement reprendre tous les labels par défaut pour les remplacer par des labels adaptés à notre type, mais également internationaliser les chaînes, afin de permettre la traduction de notre extension.

Voici notre déclaration complète de type de contenu :

```
$labels = array(
  'name'               => __('Classifieds', 'simpleclassifieds'),
  'singular_name'      => __('Classified', 'simpleclassifieds'),
  'add_new'            => __('Add new', 'simpleclassifieds'),
  'add_new_item'       => __('Add New Classified', 'simpleclassifieds'),
  'edit_item'          => __('Edit Classified', 'simpleclassifieds'),
  'new_item'           => __('New Classified', 'simpleclassifieds'),
  'view_item'          => __('View Classified', 'simpleclassifieds'),
  'search_items'       => __('Search Classifieds', 'simpleclassifieds'),
  'not_found'          => __('No classifieds found', 'simpleclassifieds'),
  'not_found_in_trash' => __('No classifieds found in trash',
'simpleclassifieds')
);
$args = array(
  'label'              => __('Classifieds', 'simpleclassifieds'),
  'labels'             => $labels,
  'public'             => true,
  'menu_position'      => 20,
  'supports'           => array( 'title', 'editor', 'author' ),
  'show_in_nav_menus' => false
);
register_post_type( 'classified', $args );
```

Notre type est donc identifié sous le nom interne "classified".

Notez que nous n'avons modifié que les options dont le réglage par défaut ne nous arrangeait pas. Par exemple, par défaut un type de contenu n'est pas public, afin de pouvoir le développer sans pour autant troubler le fonctionnement de l'interface. Grâce à `menu_position`, nous plaçons le menu de gestion des petites annonces après celui de gestion des pages. L'écran d'ajout/modification d'une annonce ne contiendra que le titre, l'éditeur de texte et la sélection d'auteur. Enfin, les annonces ne s'afficheront pas dans le menu de navigation. De nombreuses autres options sont disponibles, mais leurs valeurs par défaut nous suffisent pour le moment.

Pour assurer la traduction de toutes nos chaînes, nous les avons placées au sein de la fonction `__()`, en précisant le *text-domain* de notre extension en second argument : "simpleclassifieds".

Mise en place des taxinomies

Notre type de données est personnalisé, mais nous souhaitons tout de même que celui-ci puisse profiter des catégories et mots-clefs. Nous allons devoir recréer ces taxinomies pour le type "classified", ce qui n'est guère plus difficile que créer le type de contenu lui-même.

La fonction à utiliser cette fois est `register_taxonomy()` et, tout comme `register_post_type()`, elle suffit pour mettre en place une interface et une logique complexes mais dispose de nombre d'options.

Voici le code complet mettant en place les catégories de petites annonces, et les mots-clefs de petites annonces :

```
register_taxonomy(
  'classifieds_categories',
  'classified',
  array(
    'hierarchical' => true,
    'label'               => __('Categories', 'simpleclassifieds'),
    'labels'              => array(
      'name'              => __('Categories', 'simpleclassifieds'),
      'singular_name'     => __('Category', 'simpleclassifieds'),
      'search_items'      => __('Search Categories', 'simpleclassifieds'),
      'popular_items'     => __('Popular Categories', 'simpleclassifieds'),
      'all_items'         => __('All Categories', 'simpleclassifieds'),
      'parent_item'       => __('Parent Category', 'simpleclassifieds'),
      'parent_item_colon' => __('Parent Category:', 'simpleclassifieds'),
      'edit_item'         => __('Edit Category', 'simpleclassifieds'),
      'update_item'       => __('Update Category', 'simpleclassifieds'),
      'add_new_item'      => __('Add New Category', 'simpleclassifieds')
    ),
    'query_var'           => true,
    'rewrite'             => true
  )
);

register_taxonomy(
  'classifieds_tags',
  'classified',
  array(
    'hierarchical'        => false,
    'label'               => __('Tags', 'simpleclassifieds'),
    'labels'              => array(
      'name'              => __('Tags', 'simpleclassifieds'),
      'singular_name'     => __('Tag', 'simpleclassifieds'),
      'search_items'      => __('Search Tags', 'simpleclassifieds'),
      'popular_items'     => __('Popular Tags', 'simpleclassifieds'),
      'all_items'         => __('All Tags', 'simpleclassifieds'),
      'parent_item'       => __('Parent Tag', 'simpleclassifieds'),
      'parent_item_colon' => __('Parent Tag:', 'simpleclassifieds'),
      'edit_item'         => _('Edit Tag', 'simpleclassifieds'),
      'update_item'       => __('Update Tag', 'simpleclassifieds'),
      'add_new_item'      => _('Add New Tag', 'simpleclassifieds'),
      'separate_items_with_commas' => __('Separate tags with commas',
'simpleclassifieds'),
      'add_or_remove_items'        => __('Add or remove tags',
'simpleclassifieds'),
      'choose_from_most_used'      => __('Choose from must used tags',
'simpleclassifieds'),
    ),
    'query_var'           => true,
    'rewrite'             => true
  )
);
```