WordPress est un CMS gratuit et écrit en PHP. Il est très simple d'utilisation et peut être appréhendé par des débutants. L'ajout d'extensions est un gros point fort de WordPress. Nous allons voir dans ce TP comment créer un Plugin sur mesure. Nous considèrerons que les TP sur la prise en main de WordPress sont maîtrisés.

A faire

Dans ce TP, vous aurez besoin d'un site wordpress installé avec un accès aux fichiers du CMS.

Sommaire

- Motivations
- Déclaration d'un plugin
- Hooks
- Les actions
- Les filtres
- Les shortcodes
- Création d'un plugin de satisfaction
- Organisation du code
- Ajouter un Widget
- Déclaration du Widget
- Surcharge de la méthode widget
- Surcharge de la méthode form.
- La base de données
- L'administration

- Squelettes
- Autoformations

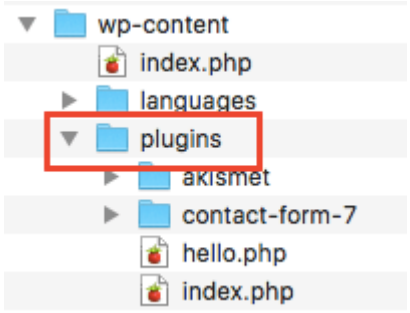
Motivations

Des milliers de Plugins gratuits existent et résolvent de nombreux problèmes. Cependant, WordPress représente aujourd'hui la solution la plus utilisée pour les sites web lien1, lien2. Des besoins personnalisés se font donc sentir. Il peut s'agir de modifier un plugin existant ou d'ajouter une idée nouvelle (commerce, jeux, sécurité, etc). Ajouter un plugin peut permettre aussi d'organiser son code et d'éviter de tout mettre dans le fichier function.php du thème utilisé.

Déclaration d'un plugin

Pour déclarer un plugin sous WordPress, il suffit de suivre la démarche suivante :

- Créer un dossier avec le nom du plugin dans le dossier racine/wp-content/plugins.



- Créer un fichier nomDuPlugin.php.
- Dans le fichier nomDuPlugin.php, écrire le code suivant en adaptant

```
<?php
 /*
 Plugin Name: monPlugin
 Plugin URI: http://monPlugin.com/
 Description: Ce plugin va révolutionner le monde.
 Version: 1.9.4
 Author: Woody Allen
 Author URI: http://monPlugin.com/
 License: GPL3
 Text Domain: monPlugin
 */
?>
```

- Le plugin est immédiatement disponible dans le backend au niveau des extensions

A faire

Créer un plugin. Activer ce plugin.

Hooks

Un hook sous WordPress est un mécanisme permettant de modifier des comportements. Il existe deux types de hooks : les actions et les filtres.

Les actions

Les actions permettent d'appeler une fonction à un moment donné. La syntaxe est

```
add_action( 'moment_de_declenchement', 'nom_de_la_fonction_de_rappel'
);
```

Beaucoup de moment sont prévus :

- 'init' pour l'initilisation
- 'get_header' avant que l'entête soit chargée
- une liste est donnée dans le codex.

Le code suivant permet d'ajouter le message Bonjour !! avant chaque page.

```
// Action Bonjour
function bonjour(){
   echo '<p> Bonjour !!</p>';
}
add_action( 'init', 'bonjour');
```

Les filtres

Les filtres modifient des éléments de WordPress. La syntaxe est :

```
add_filter( 'nom_de_lelement', 'fonction_de_modification' );
```

Les éléments modifiables sont nombreux :

- the_content fait référence au contenu d'un post.
- Les éléments modifiables sont listés dans le codex.

Le code suivant :

```
function ajoutSalut($content) {
  $content .= 'Salut';
  return $content;
}

add_filter( 'the_content', 'ajoutSalut' );
```

Cependant, les filtres doivent être chargés avant les plugins. On ne peut donc les placer dans le plugin.

Les shortcodes

Lorsqu'on crée un plugin, on utilise généralement des widgets (que nous verrons plus loin) et des shortcodes. Il s'agit de fonctions qu'il est possible d'appeler dans une page ou un article. La syntaxe est la suivante

```
[nom_du_shortcode attribut1="valeur1" attribut2="valeur2"]
```

ou en ajoutant du texte comme avec des balises HTML

```
[nom_du_shortcode]le contenu du texte[/nom_du_shortcode]
```

Par exemple, le code suivant permet d'ajouter une galerie d'images, deux colonnes, taille moyenne.

```
[*gallery columns="2" size="medium" ]
```

Le short code su_private permet de cacher un texte aux visiteurs non connectés.

```
[*su_private]Texte privé[*/su_private]
```

Pour créer un shotcode, il suffit d'ajouter le code suivant:

```
function fonctionGrandTitre($param, $content) {
   return '<h1>' . $content . '</h1>';
}
add_shortcode('grandTitre', 'fonctionGrandTitre');
```

Ce code comprend la fonction appelée et la déclaration du shortcode.

Pour utiliser ce code, il suffit d'écrire dans une page ou un article

```
[grandTitre] Super Titre !!! [/grandTitre]
```

Pour utiliser des paramètres, il faut utiliser le code suivant :

```
function fonctionNomTaille($param) {
  extract(
    shortcode_atts(
      array(
        'nom' => 'Dupont',
        'taille' => 186
      ),
      $param
    )
  );
  return 'Monsieur '.$nom." mesure ".$taille;
);
add_shortcode('nomTaille', 'fonctionNomTaille');
```

Des valeurs par défaut sont données si les attributs ne sont pas remplis. L'utilisation se fait de la manière suivante :

```
[nomTaille nom="Paul" taille="165"]
```

Création d'un plugin de satisfaction

Nous allons maintenant créér un plugin qui permettra à tous les visiteurs de dire s'ils aiment le site ou non. Un shortcode permettra d'afficher les résultats et l'administrateur, pourra réinitialiser le sondage ou tricher. Nous allons organiser notre code en objet afin de prendre de bonnes habitudes.

A faire

Suivre les instructions qui suivent pour réaliser le plugin de satisfaction.

Dans ce TP, nous mettrons l'essentiel du code dans un seul fichier. Cependant, dans un projet réel, il vaudrait mieux séparer les différentes parties du code. Des constantes sont prévues pour les chemins. La commande

```
plugin_dir_path( __FILE__ )
```

Permet d'obtenir le chemin absolu vers le dossier du plugin. Ainsi, pour inclure un fichier

nommé monFichier.php et situé à la racine du dossier de plugin, il suffit d'utiliser la commande :

```
include_once plugin_dir_path( __FILE__ ).'/monFichier.php';
```

Organisation du code

Nous allons créer une classe Satisfaction afin de profiter des facilités de la programmation orientée objet.

```
class Satisfaction
{
  public function __construct()
  {
    // des trucs a mettre
  }
}

new Satisfaction();
```

Des méthodes de cette classe vont être appelées à différents moments prévus par WordPress. Nous allons définir des fonctions qui doivent s'executer à l'activation, la désactivation et la désinstallation du plugin.

```
class Satisfaction
{
  public function __construct()
  {
    // des trucs a mettre
  }
  public static function install(){

  }
  public static function uninstall(){

  }
  public static function desactivate(){

  }
}

new Satisfaction();
```

Afin d'appeler ces fonctions au bon moment, nous allons utiliser des hooks. La commande suivante utilise le hook register_activation_hook qui est appelé au moment de l'activation du plugin. Les paramètres qui suivent permettent de localiser la fonction à appeler. Notons la syntaxe particulière dans de cas, puisque pour appeler la méthode install de la classe Satisfaction il faut utiliser un tableau avec le nom de la classe et le nom de la méthode.

```
```

```
register_activation_hook(__FILE__,array('Satisfaction','install'));
```

De même, pour la désactivation et la désinstallation du plugin

```
register_activation_hook(__FILE__,array('Satisfaction','install'));
register_deactivation_hook(__FILE__,array('Satisfaction','desactivate'
));
register_uninstall_hook(__FILE__,array('Satisfaction','uninstall'));
```

Ajouter un Widget

Pour créer un widget dans WordPress, il faut créer une classe qui hérite de la classe WP_Widget. La surcharge de différentes méthodes permettra la construction de notre widget.

```
class afficheQuestion extends WP_Widget
{
  public function __construct()
  {
    parent::__construct('idAfficheQuestion', 'Affiche Question',
array('description' =&amp;amp;amp;amp;amp;amp;amp;gt; 'Un formulaire
pour répondre'));

  }
```

```
}
```

Le constructeur de la classe WP_Widget prend en paramètres, un identifiant pour le widget, ce qui sera affiché dans l'administration et des options dans un tableau, ici la description. La totalité des possibilités est décrite dans le codex.

Déclaration du Widget

Il faut maintenant déclarer le plugin au moment où le plugin est chargé. Pour cela nous allons le hook qui se déclenche au moment où on charge les widgets, à savoir add_action ('widgets_init',... et appeler une fonction qui déclare notre widget. La fonction à ajouter est

```
function declarerWidget(){
register_widget('afficheQuestion');
}
```

Puis, dans le constructeur de notre classe Satisfaction, nous ajoutons le hook

```
add_action('widgets_init','declarerWidget');
```

Le widget est alors déclaré.

Surcharge de la méthode widget

L'affichage de notre widget dans le Front-End sera réalisé en surchargeant la méthode widget.

Nous allons créer un formulaire en ajoutant la méthode suivante à la classe afficheQuestion.

```
public function widget(){
  echo "<form action="" method="post">
    <h1>Aimez-vous ce site ?</h1>
     <input type="checkbox" name="oui" id="oui"/> Oui
     <input type="checkbox" name="non" id="non"/> Non <br/>
    <input type="submit"/>
  </form>";

}
```

Surcharge de la méthode form.

Le paramétrage de notre widget dans le Back-End se fera en surchargeant la méthode form.

Nous allons maintenant ajouter la possibilité de modifier la question du formulaire. Pour créer une série de sondages différents avec différentes réponses, il faudrait procéder différemment mais nous n'avons pas encore les outils. Commençons simplement, nous pourrons toujours compliquer l'affaire ensuite.

Nous allons donc surcharger la méthode form.

```
public function form($instance){
   $question =
isset($instance['question'])?$instance['question']:'Aimez-vous ce site
?';
   $id = $this->get_field_id('question');
   $name = $this->get_field_name('question');
   echo "<p>Nom de la question <input type='text' id=$id
name='".$name."' value='".$question."'></p>";
 }
```

La variable $instance permet de récupérer toutes les instances créées pour ce widget. Dans la variable $question nous récupérons la valeur de l'instance question si elle existe sinon, on lui affecte une valeur par défaut. Les méthodes get_field_id et get_field_name permettent de récupérer identifiant et le nom de l'instance question.

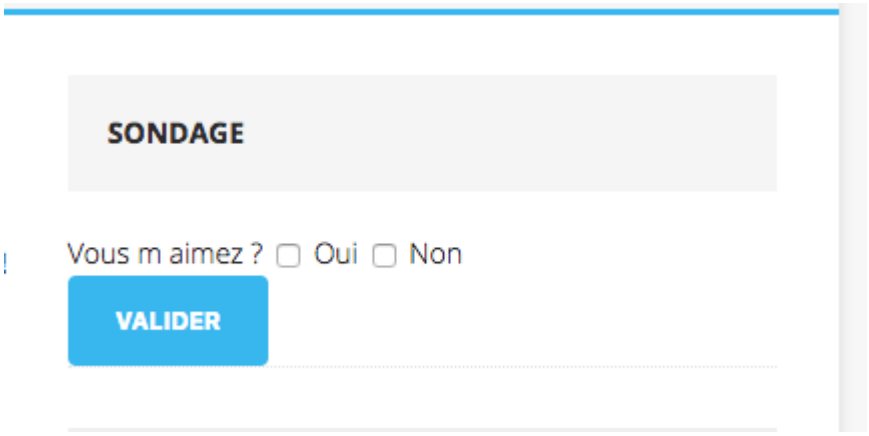
Du coup, on peut remplacer le texte dans la méthode widget par la valeur de l'instance question.

```
public function widget($args,$instance){
  echo "<form action="" method="post">
    <h1>".$instance['question']."</h1>
     <input type="checkbox" name="oui" id="oui"/> Oui
     <input type="checkbox" name="non" id="non"/> Non <br/>
    <input type="submit"/>
  </form>";

}
```

Il est possible également d'appliquer un filtre de manière à pouvoir modifier les widgets d'un thème simplement

```
public function widget($args,$instance){
  $question = apply_filters('widget_text', $instance['question']);
  echo "<form action="" method="post">
    <h1>".$question."</h1>
     <input type="checkbox" name="oui" id="oui"/> Oui
     <input type="checkbox" name="non" id="non"/> Non <br/>
    <input type="submit"/>
  </form>";

}
```

Le hook widget_text va s'appliquer à tous les textes des widgets. Comme d'habitude la liste se trouve dans le codex.



La base de données

Nous allons maintenant enregistrer le résultat des votes. La base de données dans wordpress est gérée de manière très simple. Tout est dans la variable $wpdb (wordpress database). La méthode query permet de réaliser une requête. Nous allons créer une fonction install_db dans la classe Satisfaction.

```
public function install_db(){

  global $wpdb;

  $wpdb->query("CREATE TABLE IF NOT EXISTS
".$wpdb->prefix."reponse_question (id int(11) AUTO_INCREMENT PRIMARY
KEY, reponse tinyint(1), idUser int(11));");

}
```

La commande $wpdb->prefix permet d'obtenir le prefix de la base de données wordpress. On crée un base de données avec une requête en SQL. Il suffit ensuite d'ajouter la commande

```
Satisfaction::install_db();
```

dans la méthode install de la classe Satisfaction.

Il faut également supprimer la table lorsqu'on désinstalle le plugin. On place alors une fonction un_install_db() dans la classe Satisfaction.

```
public function uninstall_db(){

  global $wpdb;

  $wpdb->query("DROP TABLE IF EXISTS
".$wpdb->prefix."reponse_question;");

}
```

On place alors la commande

```
Satisfaction::uninstall_db();
```

dans la méthode uninstall de la classe Satisfaction.

Il nous reste maintenant à voir l'insertion des votes dans la base de donnée et la sélection pour l'affichage des résultats.

Pour insérer des éléments dans la base de données, il faut utiliser la méthode insert. Dans la méthode widget de la classe afficheQuestion ajoutons le code suivant :

```
global $wpdb;

$table =$wpdb->prefix.'reponse_question';
$idUser = get_current_user_id();
if(isset($_POST['oui'])){
  $wpdb->insert( $table, array('idUser'=>$idUser,'reponse'=>1));
}
if(isset($_POST['non'])){
  $wpdb->insert( $table, array('idUser'=>$idUser,'reponse'=>0));
}
```

Nous allons maintenant ajouter un shortcode afin de visualiser le résultat de la question. Ajoutons à la classe Satisfaction la méthode suivante :

```
public static function resume($param){
 global $wpdb;
 $query = "SELECT * FROM ".$wpdb->prefix."reponse_question";
```

```
  $resultats = $wpdb->get_results($query) ;
  $oui = 0;
  $non = 0;
  foreach($resultats as $rep){
    if($rep->reponse==1)
      $oui++;
    else
      $non++;
    }
  return "Oui : $oui, Non : $non";
}
```

Dans le code précédent, on commence par récupérer l'instance de la base de donnée. Ensuite, on crée la requête SQL. La méthode get_results permet de récupérer les enregistrements. Dans le foreach chaque enregistrement est un objet dont chaque champ est un attribut. Ici, on cherche à récupérer une réponse dont il faut utiliser l'attribut reponse. C'est pourquoi on utilise la commande $rep->reponse.

Insertion du shortcode [resume]

Résultat du shotcode [resume]

L'administration

Tout bon plugin inclut une gestion dans le Back-End (l'administration).

Il ne nous reste plus qu'à ajouter à notre plugin la possibilité d'être réinitialisé ou d'être

trafiqué par l'administrateur. Pour ajouter des éléments au menu d'administration, il faut utiliser les fonctions add_menu_page, add_submenu_page, add_option_page, etc. Ces fonctions sont disponibles sur le codex et par exemple ici. Comme précédemment, il faut charger le menu au moment opportun. Nous allons donc une nouvelle fois utiliser un hook. Ajoutons au constructeur de la classe Satisfaction la ligne de commande suivante :

```
add_action('admin_menu',array($this,'declareAdmin'));
```

Dans ce code, nous utilisons le hook admin_menu qui appelle la méthode declareAdmin (par encore écrite) de la classe Satisfaction au moment de la déclaration des menus du Back-End. Créons maintenant la dite méthode en ajoutant le code suivant à la classe Satisfaction.

```
public static function declareAdmin(){
add_menu_page('Configuration du questionnaire', 'Questionnaire',
'manage_options', 'question', array($this, 'menuHtml'));
}
```

- Le premier paramètre représente le titre de la page sur laquelle nous seront dirigés en cliquant sur le menu. Il sera récupéré grâce à la fonction get_admin_page_title().
- Le deuxième paramètre sera le texte affiché dans le menu.
- Le troisième paramètre donne les droits que doit posséder l'utilisateur pour accéder à ce menu.
- Le quatrième paramètre est un mot-clé pour faire référence au menu.
- Enfin, le dernier paramètre indique la fonction qui va gérer l'affichage du menu (pas encore

créée).

Notons que pour utiliser la méthode d'une classe il est possible d'instancier la classe, par exemple $truc = new maClasse();. Dans ce cas, il faut utiliser la syntaxe suivante :

```
array(&$truc,'menuHtml');
```

Il faut maintenant créer la méthode menuHtml. Insérons la méthode suivante dans la classe Satisfaction.

```
public static function menuHtml(){
  echo '<h1>'.get_admin_page_title().'</h1>';
  echo '<p>Page du plugin Questionnaire !!!</p>';
}
```

La fonction menuHtml gère l'affichage de la page d'administration du plugin.

Créons deux sous menus, le premier pour réinitialiser le sondage et l'autre pour tricher.

Ajoutons le code suivant à la méthode declareAdmin de la classe Satisfaction.

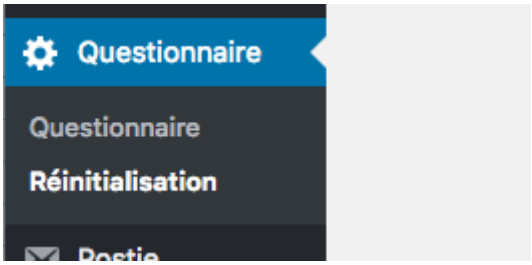
```
add_submenu_page('question','Réinitialisation du
questionnaire','Réinitialisation','manage_options','reinit',array($thi
s,'menuHtmlInit'));
```

Le premier paramètre représente le menu parent.

Il faut ensuite ajouter la méthode menuHtmlInit à la classe Satisfaction.

```
public static function menuHtmlInit(){
 global $wpdb;
 echo '<h1>Réinitialisation</h1>';
 echo '<p>Cliquer sur le bouton suivant pour réinitialiser le
questionnaire</p>';
 echo "<form method='POST' action='#'>
 <input type='submit' name='reinit'>
 </form>
 ";
 if(isset($_POST['reinit'])){
   $query = 'TRUNCATE TABLE '.$wpdb->prefix.'reponse_question';
   $wpdb->query($query);
   echo "réinitialisation !!!";
 }

 }
```

Il s’agit d’un simple formulaire et de l’utilisation de l’instanciation $wpdb.



Sous-menu de réinitialisation

Page pour réinitialiser le questionnaire

A faire

Le sous-menu de triche est laissé en exercice en TP !!! A vous de jouer !

Squelettes

Les plugins et les widgets ont des modèles prêts à l'emploi sur internet. Voici quelques exemples. Il sont bien structurés mais il est toujours utile d'avoir son propre module prêt à l'emploi:

- BoilerPlate
- wp-boilerplate
- Widget boilerplate

Autoformations

De nombreux sites sont dédiés à la création de plugin sous wordpress. Je me suis inspiré de ces sites pour réaliser ce TP. Citons

- OpenClassRoom, très bon tuto, ce que j'ai trouvé de mieux.
- https://www.hostinger.fr/tutoriels/creer-un-plugin-wordpress/ assez simple mais efficace.
- SupInfo