PRojet plugin wordpress de vente de billets en ligne

Rapport de projet





[Introduction 2](#_Toc420346245)

[Client 2](#_Toc420346246)

[Développeur 2](#_Toc420346247)

[Superviseur 2](#_Toc420346248)

[Projet 2](#_Toc420346249)

[Cachier des charges 3](#_Toc420346250)

[Préparation 4](#_Toc420346251)

[Analyse du projet 4](#_Toc420346252)

[Planification 5](#_Toc420346253)

[Base de données 6](#_Toc420346254)

[Analyse 6](#_Toc420346255)

[Schéma 6](#_Toc420346256)

[Prototype de la partie administrative du plugin dans WordPress 7](#_Toc420346257)

[Analyse 7](#_Toc420346258)

[Réalisation 7](#_Toc420346259)

[Structo-code 10](#_Toc420346260)

[Installation du site web 11](#_Toc420346261)

[Installation du site 11](#_Toc420346262)

[Importation des données fournies par le client 13](#_Toc420346263)

[Thème 13](#_Toc420346264)

[Pages et articles 14](#_Toc420346265)

[Widget 15](#_Toc420346266)

[Création du plugin 16](#_Toc420346267)

[Préparation 16](#_Toc420346268)

[Réalisation 16](#_Toc420346269)

[Debugging 16](#_Toc420346270)

[Base du plugin 17](#_Toc420346271)

[Base menu backend 18](#_Toc420346272)

[Création des tables 20](#_Toc420346273)

[Administration 25](#_Toc420346274)

[Lancement du plugin 36](#_Toc420346275)

[Conclusion 37](#_Toc420346276)

[Annexes 38](#_Toc420346277)

# Introduction

## Client

Nom: Clavien

Prénom: Tony

Téléphone: +41 79 380 77 50

Mail: tony.clavien@japan-impact.ch

## Développeur

Nom: Mottier

Prénom: Fabien

Téléphone: +41 78 826 02 92

Mail: mottierfabien@gmail.com

## Superviseur

Nom: Savioz

Prénom: Patrick

Mail: patrick.savioz@eptsion.ch

## Projet

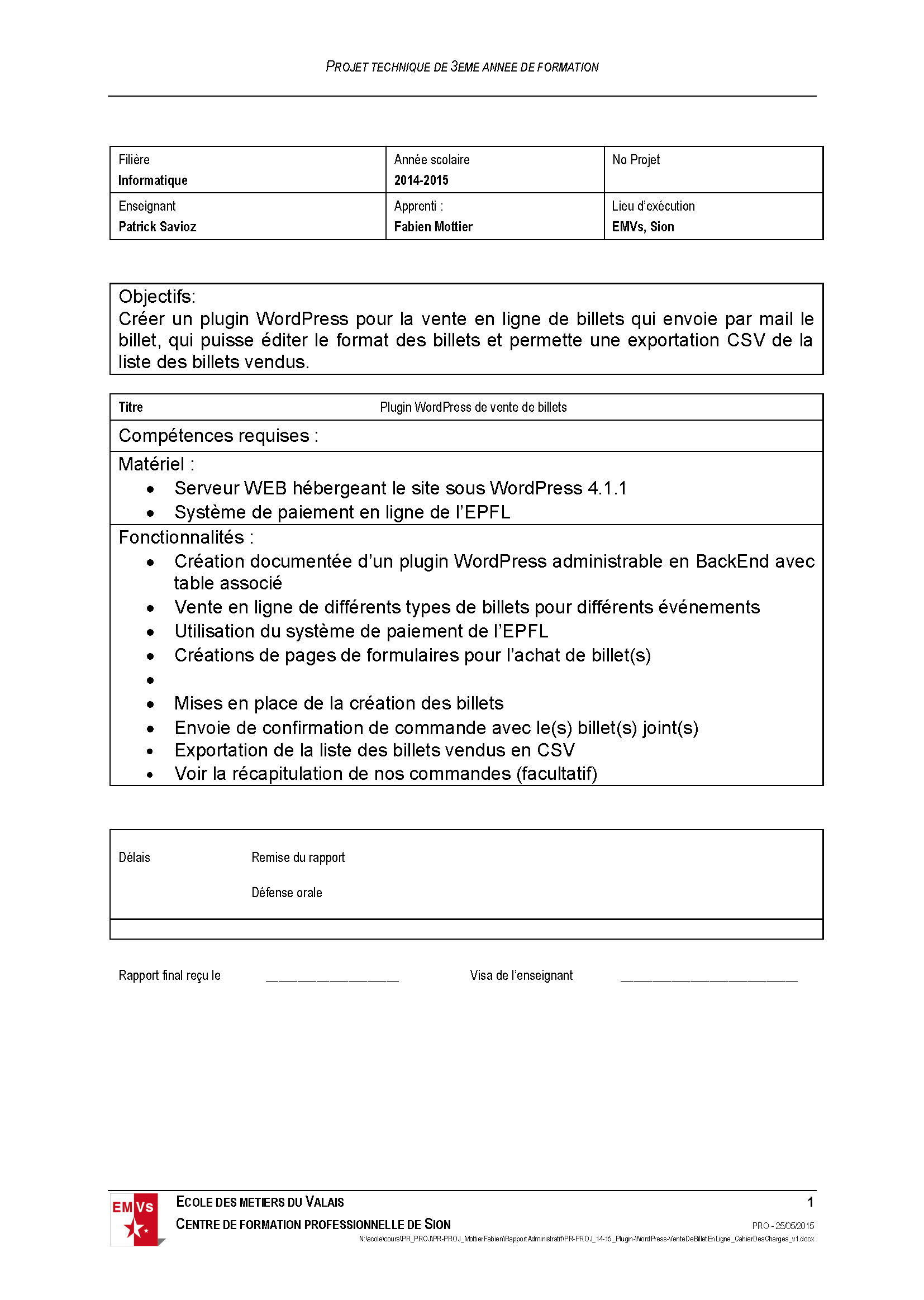
Module: PR\_PROJ

J’ai été chargé par un des organisateurs de la Japan Impact, monsieur Tony Clavien, de faire un plugin pour la vente en ligne de billets adaptables pour différents événements.

Ce plugin va générer, par le biais d’un raccourci, un formulaire pour, soit effectué une commande de différents billets, soit voir le récapitulatif de nos commandes. Si on choisit de voir le récapitulatif, on est redirigé sur une page de connexion, puis sur la page de récapitulation. Si on effectue une commande, on arrive sur une page qui nous permet soit de nous inscrire, soit de nous connecter. Après s’être connecté, on est redirigé sur l’API fournit par l’EPFL, POLYBANKING, qui va gérer le paiement en ligne. Une fois le paiement effectué, on est redirigé sur une page de confirmation de commandes avec la liste des billets à télécharger, que l’on reçoit aussi par mail, et avec un lien pour accéder à la page de récapitulation. Le plugin doit être capable de gérer plusieurs événements en même temps.

L’architecture du plugin est schématisée plus loin à la *Figure* 1.

## Cachier des charges



# Préparation

## Analyse du projet

Je me suis tout d’abord documenté sur la création de plugin WordPress grâce au site :  
 [http://openclassrooms.com/courses/propulsez-votre-site-avec-WordPress](http://openclassrooms.com/courses/propulsez-votre-site-avec-wordpress)

Ensuite j’ai eu une séance avec le client. Il m’a expliqué en détail ce dont il avait besoin et montré les prototypes de ce qu’il attendait comme plugin:

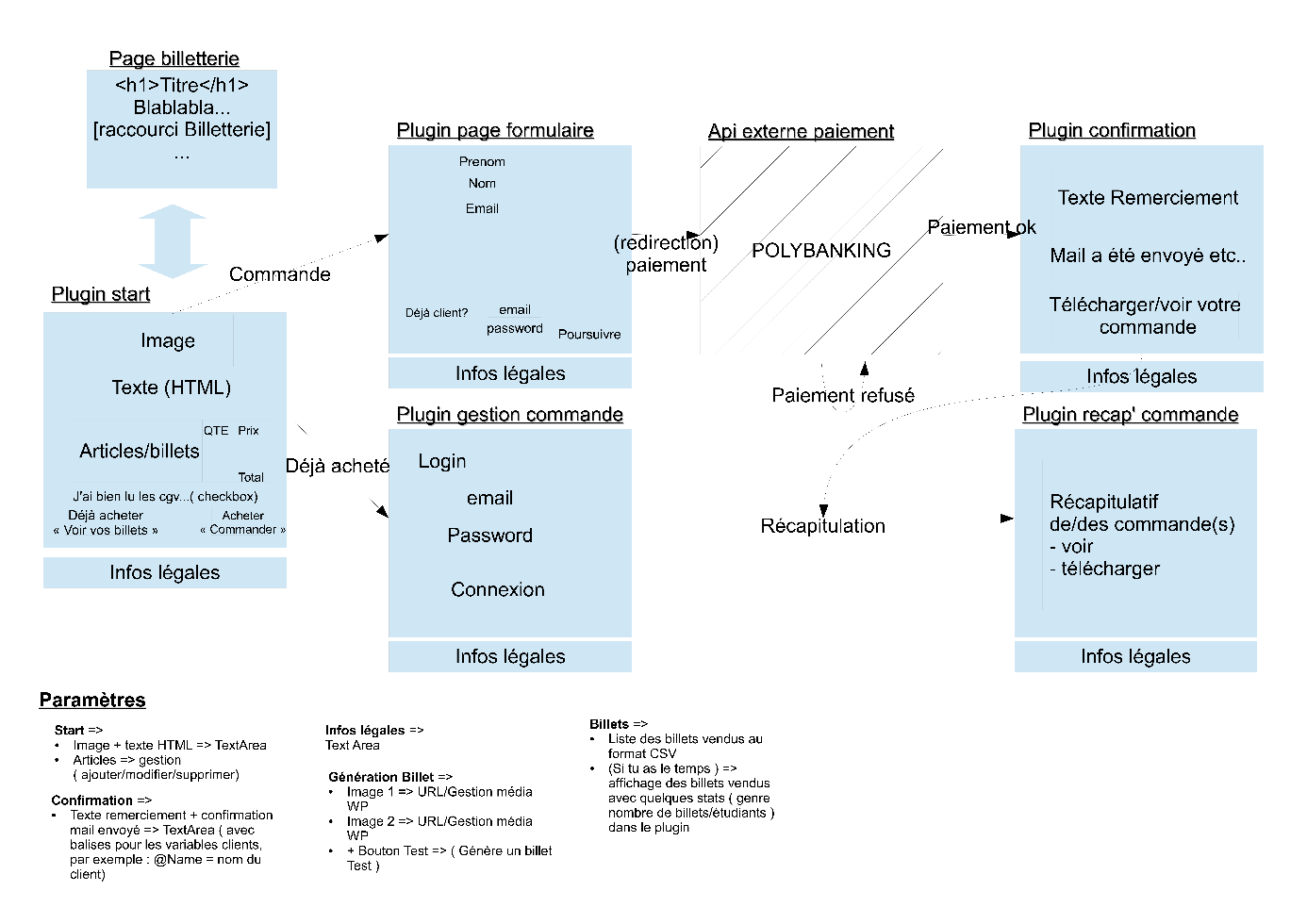


Figure 1 - Prototype général des fonctionnalités du plugin

Ce prototype montre l’architecture de la section de vente de billets sur le site web.

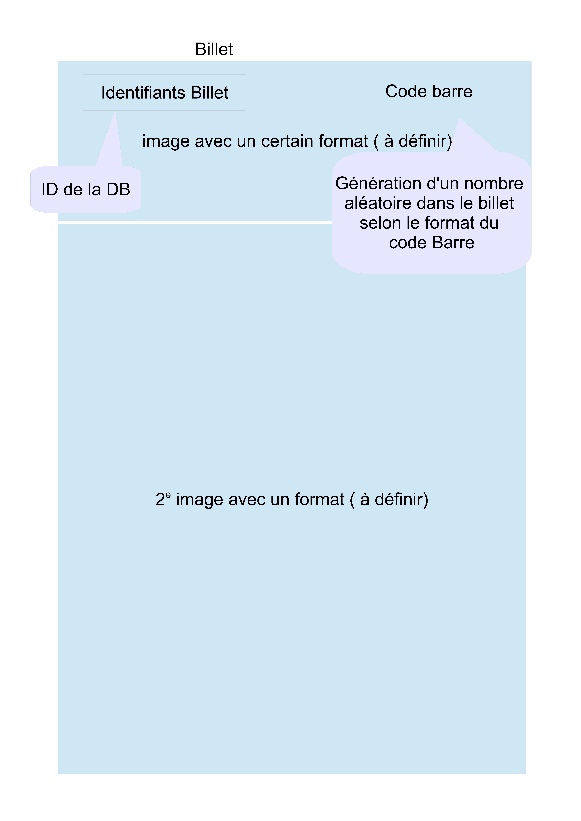
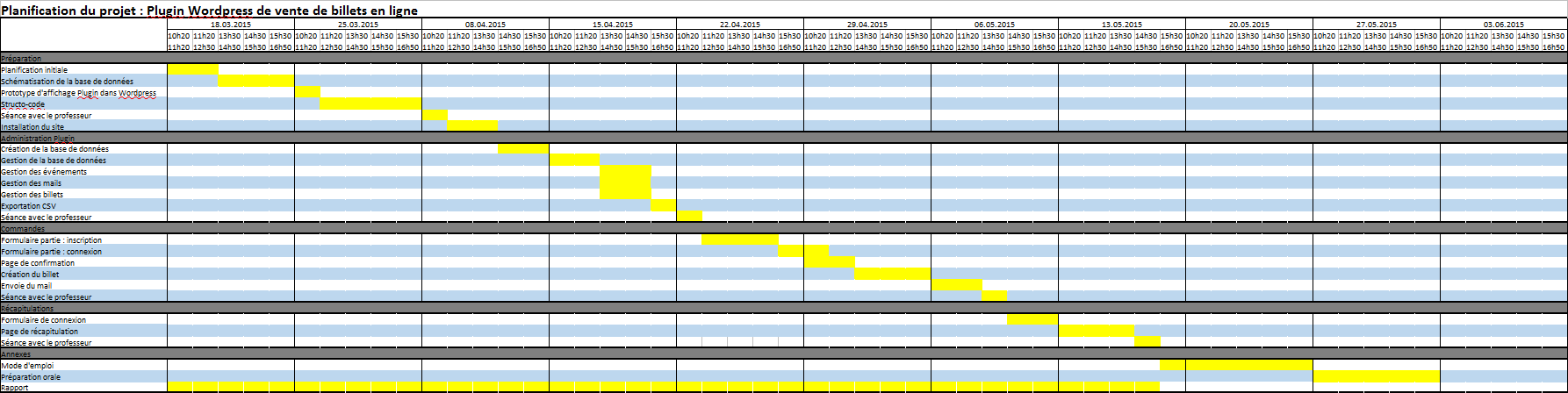


Figure 2 - Prototype d'un billet

Ceci est le prototype d’un billet vendu. Le client a souhaité deux zones d’insertion d’images pour mettre les éléments dont il avait besoin. Les informations des identifiants et du code barre seront fournis par le plugin.

## Planification



## Base de données

### Analyse

Je dois mettre en place une base de données qui va permettre la gestion de login des utilisateurs, la gestion d’événements ainsi que la gestion de commandes de billets. Certains éléments non-essentiels et qu’on veut pouvoir modifier rapidement dans le back end de WordPress comme les images pour le PDF du billet ne sont pas mis dans ce schéma, car ils se trouveront dans une table propre à WordPress.

Pour les utilisateurs, le client n’a pas besoin de beaucoup d’informations, une table avec prénom, nom, mot de passe et adresse mail suffit.

Pour les événements, la table événement contiendra le nom et l’édition pour les événements qui sont par exemple annuels. Ainsi on pourra avoir un Japan Impact 07 et un Japan Impact 08.

Pour la gestion, des billets, il y aura une table commande avec la date et le client et une table de billet avec la commande, le type de billet, dans une autre table avec le prix, le code barre généré et le PDF final du billet.

### Schéma

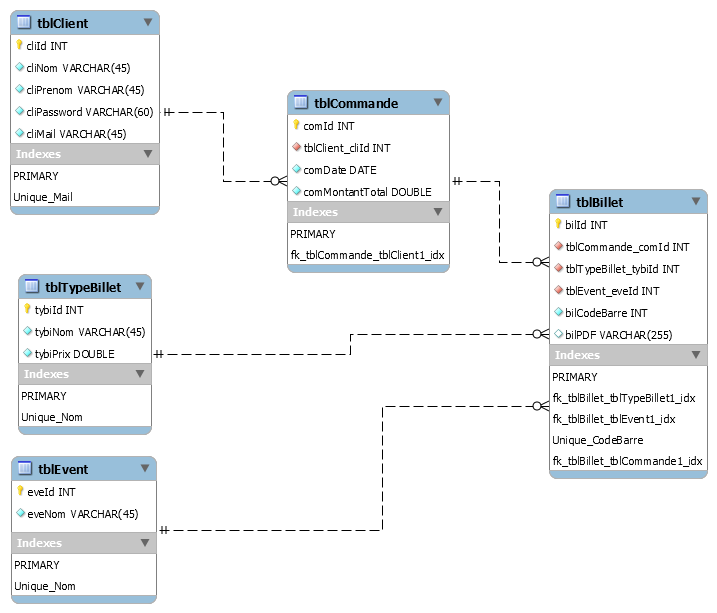


Figure 3 - Schéma de la base de données

## Prototype de la partie administrative du plugin dans WordPress

### Analyse

Je me suis documenté sur l'administration de plugin sur le site :   
[http://openclassrooms.com/courses/propulsez-votre-site-avec-WordPress/l-administration-2](http://openclassrooms.com/courses/propulsez-votre-site-avec-wordpress/l-administration-2)

J’ai ainsi découvert qu’on peut utiliser des options déjà créées ou en créer nous-mêmes. Les options sont des réglages accessibles dans le back end de WordPress qui nous permettent de modifier des éléments rapidement et facilement. Ces options enregistrent les données dans la table wp\_options qui est propre à WordPress.

J'ai utilisé le logiciel gratuit Pencil 2.0.5.

Pour la structure de l’administration du plugin, j’ai choisis de mettre en place un sous-menu pour la gestion de la base de données et un sous-menu par événement. La gestion d’événement se fait en trois partie : celle de l’événement lui-même ; celle des mails liés à celui-ci ; celle des pages liées à celui-ci.

### Réalisation

Lorsqu'un événement est créé, le plugin crée automatiquement une page accessible par un sous-menu, dans le back end de WordPress.

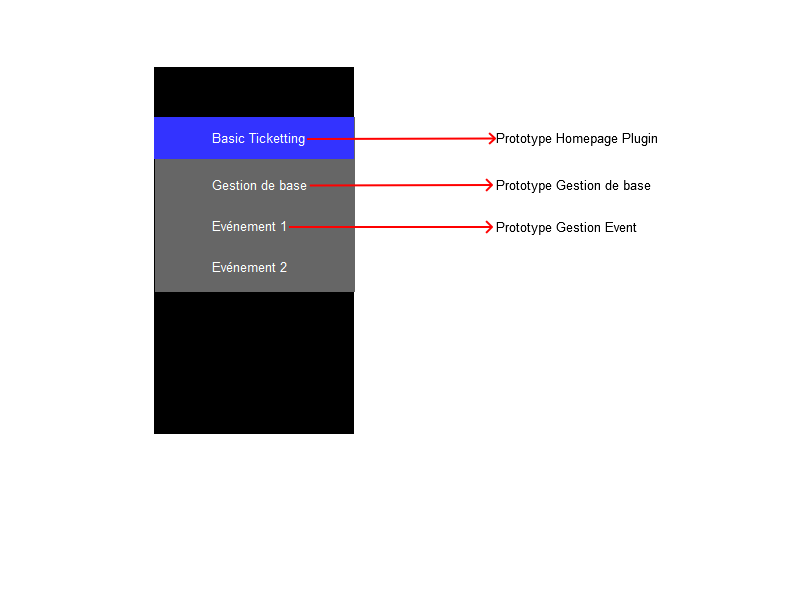


Figure 4 - Onglet dans le menu de WordPress

La homepage est une page de présentation du plugin avec un lien pour accéder au manuel.

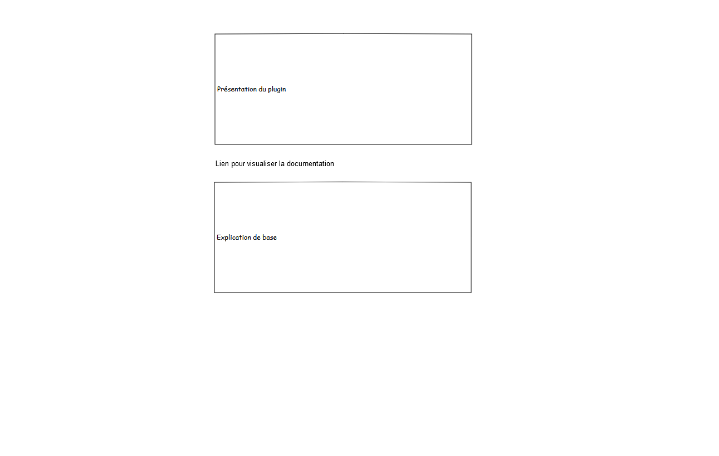


Figure 5 - Prototype de la home page

Il y aura un sous-menu de base de gestion de la base de données pour ajouter des événements et des types de billets.

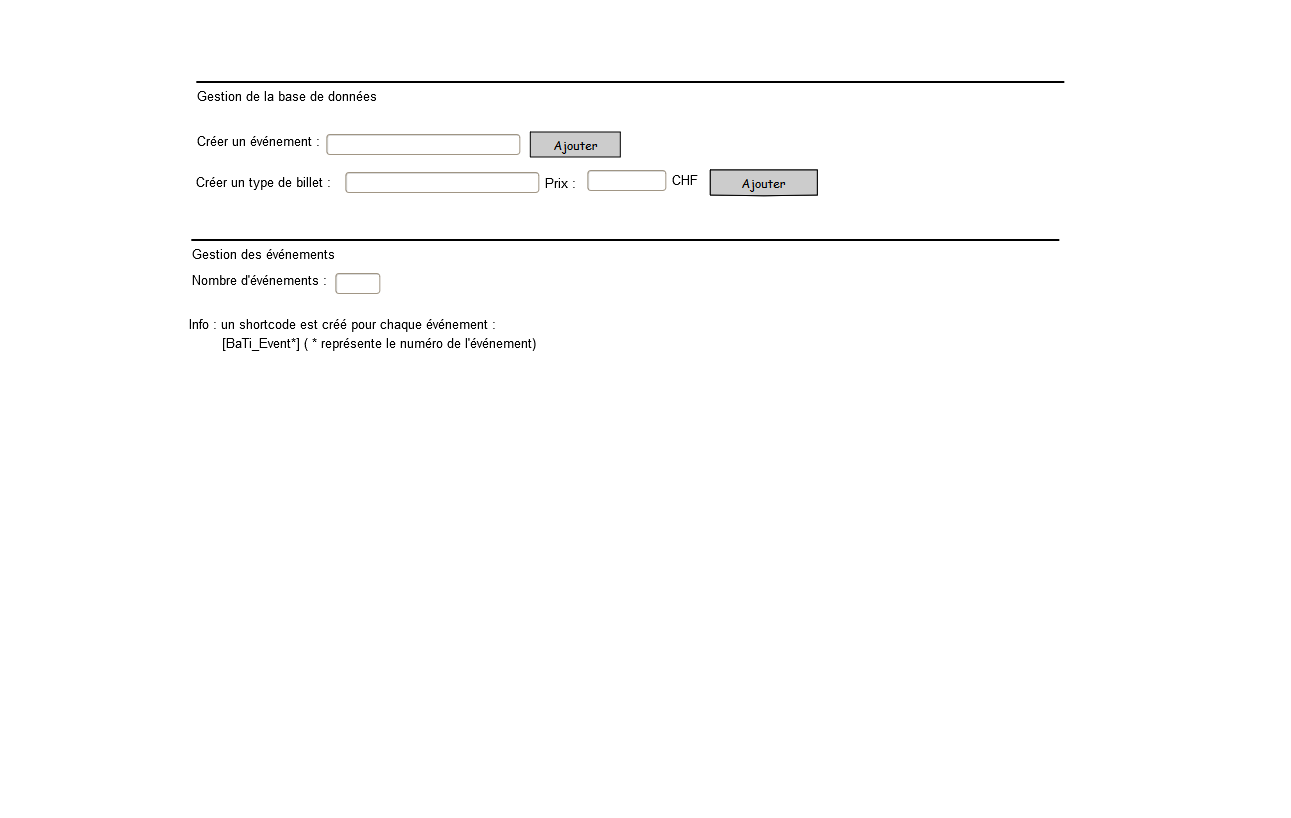


Figure 6 - Prototype de la section de gestion de la base de données

La gestion des événements sont répartis en trois onglets. L'onglet "Event" permet de choisir l’événement actuel, ses types de billets et les images pour les billets.

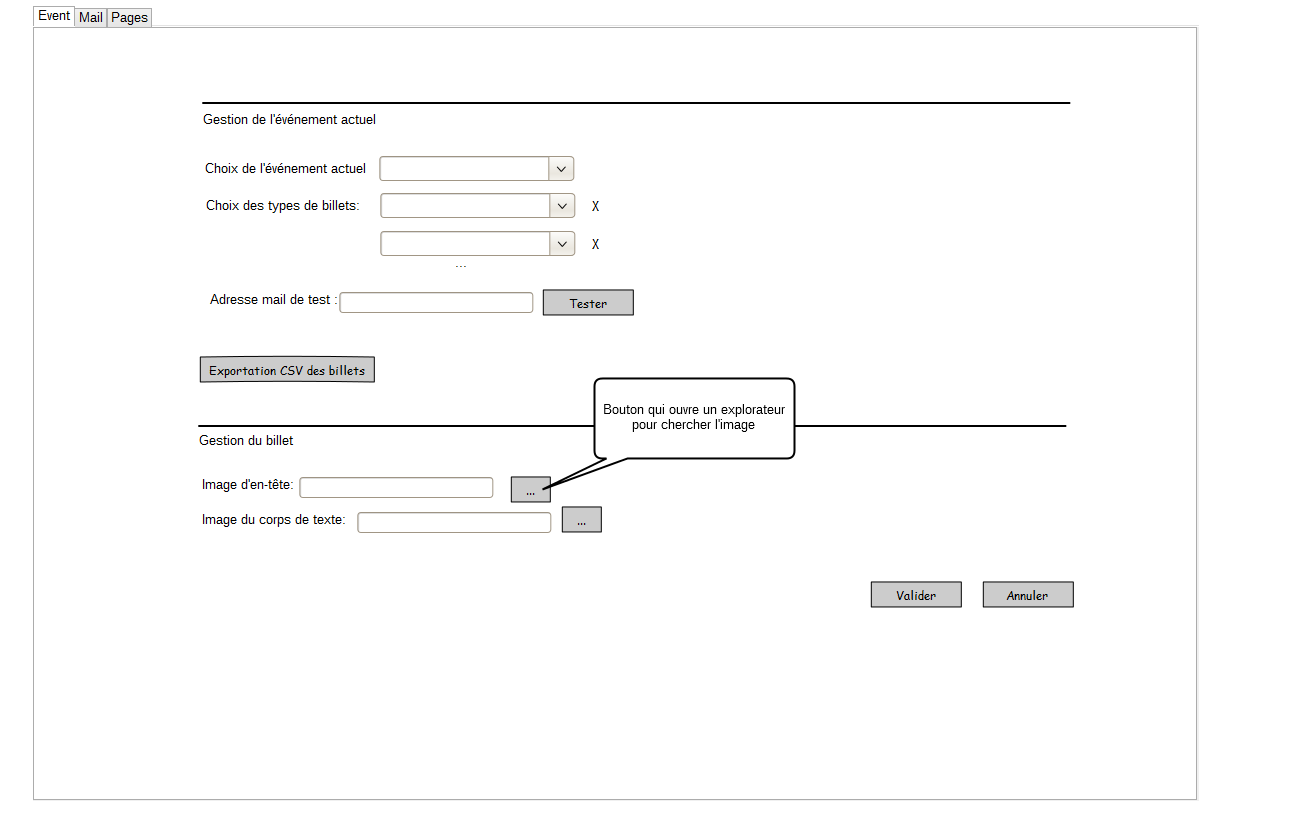
**

Figure 7 - Prototype de la section de gestion de l’évément

L'onglet "Mail" concerne les réglages pour les mails de confirmation d'inscription et de commande. Il permet d'introduire du texte dans les deux types de mail, ainsi que de rajouter des pièces jointes au mail de confirmation de commande. Il y a aussi un réglage optionnel pour utiliser une adresse mail spécifique pour l'expéditeur des mails.

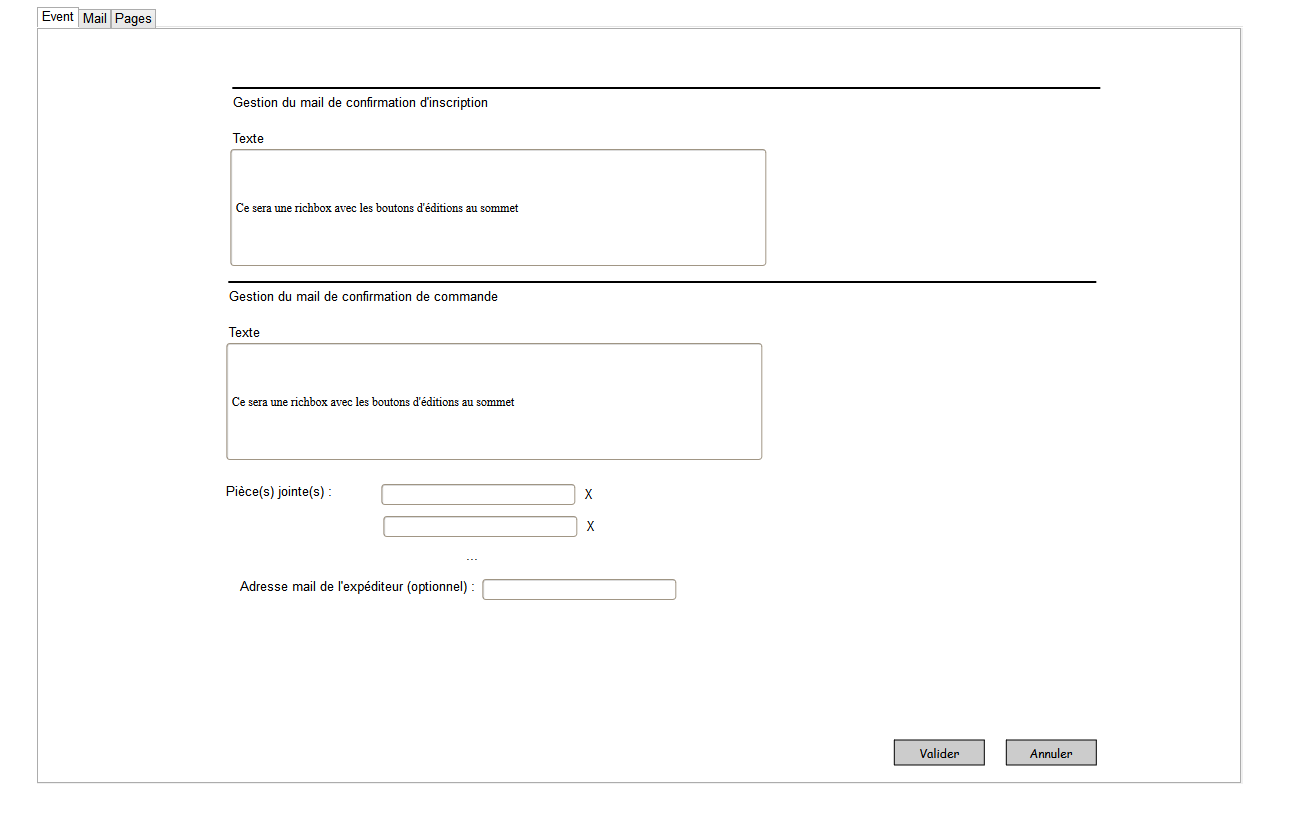


Figure 8 - Prototype de la section des mails

L'onglet "Pages" concerne les informations modifiables sur les pages créées par le plugin. Il y a la partie des informations légales qui se retrouvera sur chaque page, puis une image et du texte sur la page de démarrage du plugin et une autre zone de texte pour la page de confirmation de commande.

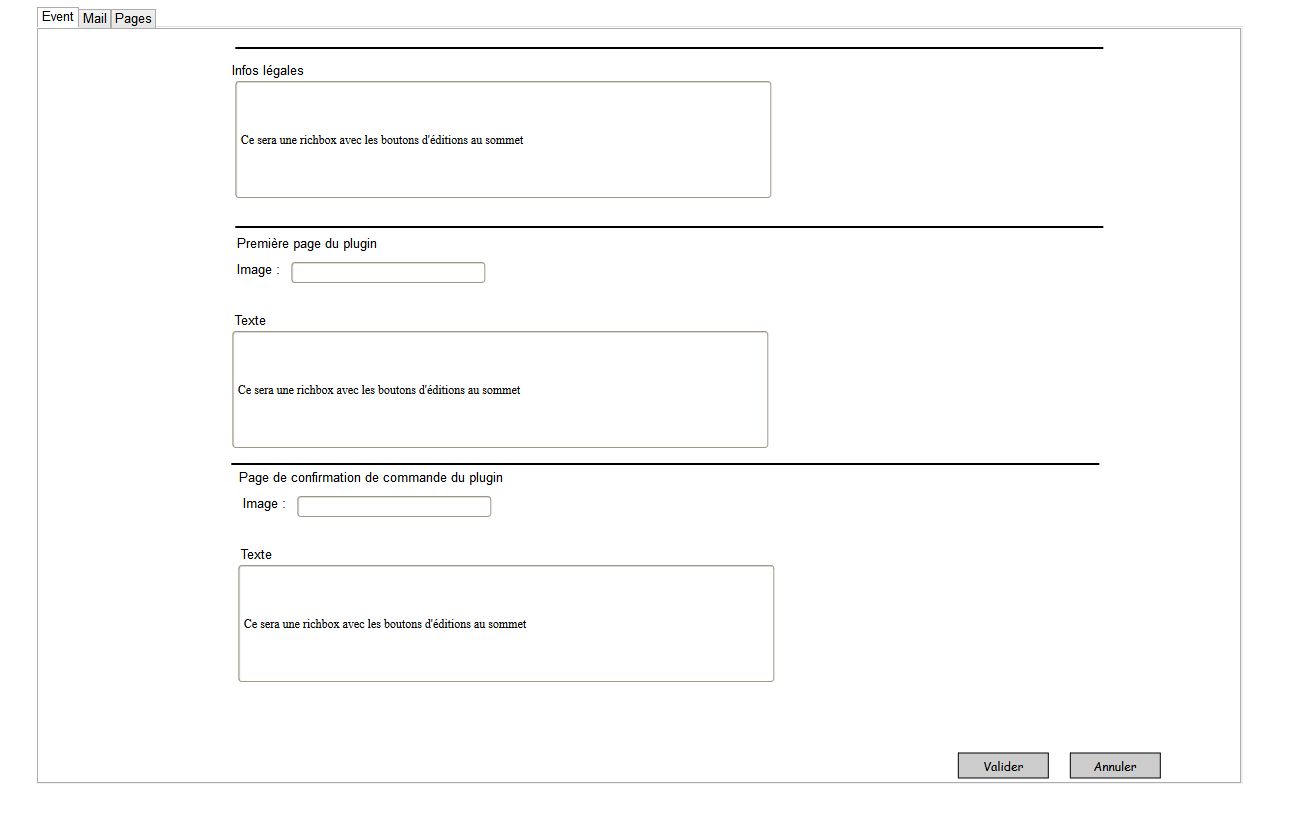


Figure 9 - Prototype de la section des pages

## Structo-code

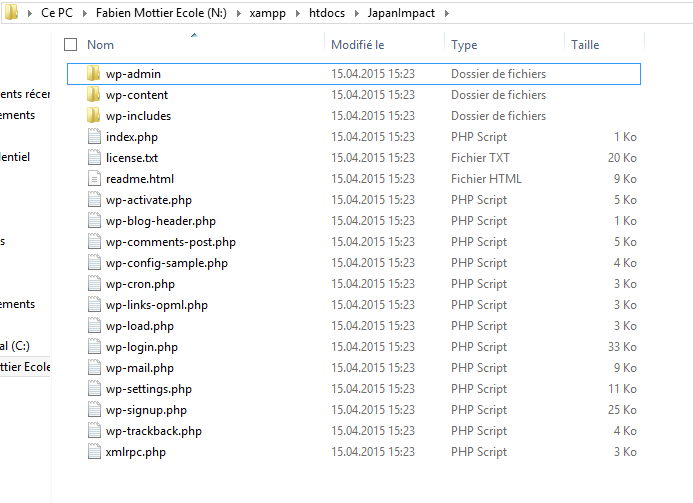
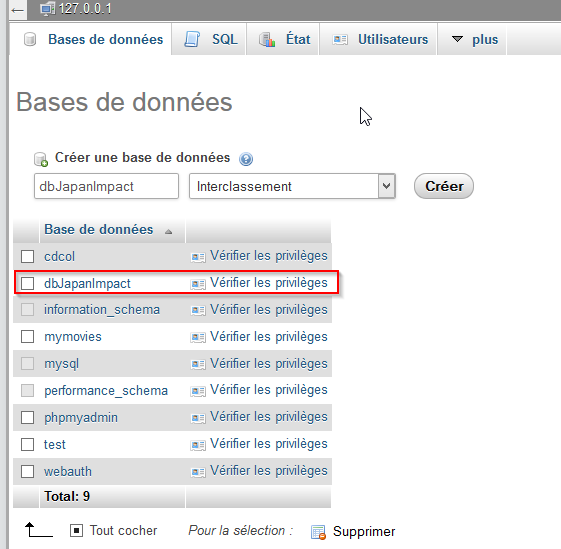
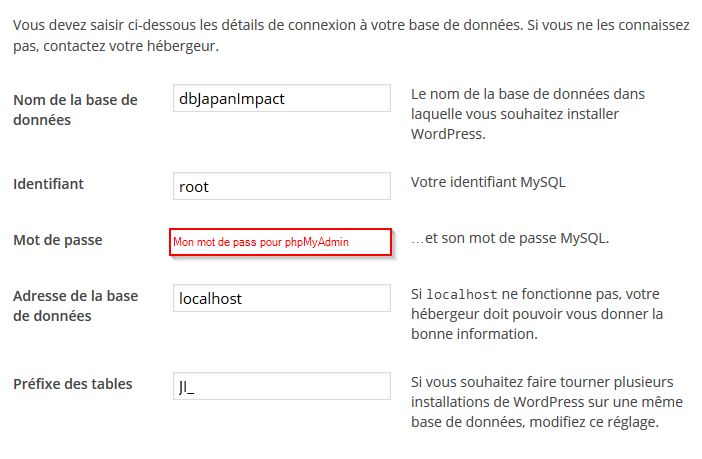
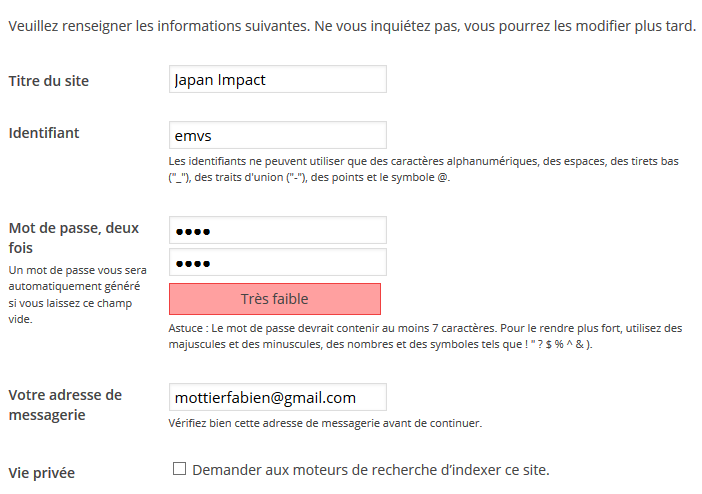
J’ai pris trop de retard et découvert que le 08 avril je n’avais pas de cours. A cause de cette perte de temps trop importante, j’ai décidé d’annuler cette partie.

# Installation du site web

J’ai choisis d’installer une réplique du site web du client, en l’occurrence http://www.japan-impact.ch/, pour avoir un cas concret.

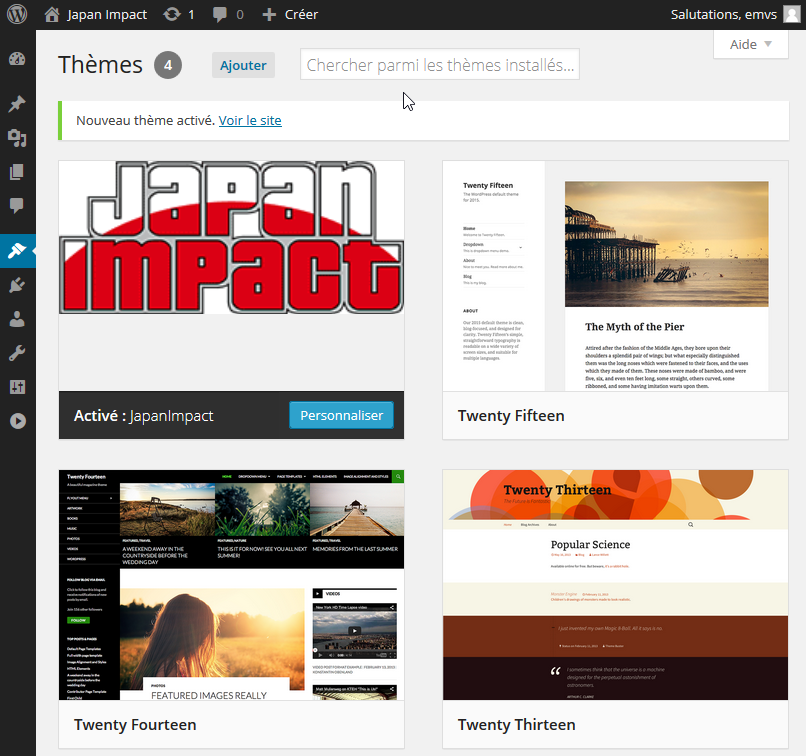
J’ai utilisé XAMPP comme serveur web. Pour WordPress, j’ai utilisé la version 4.1.1.

## Installation du site

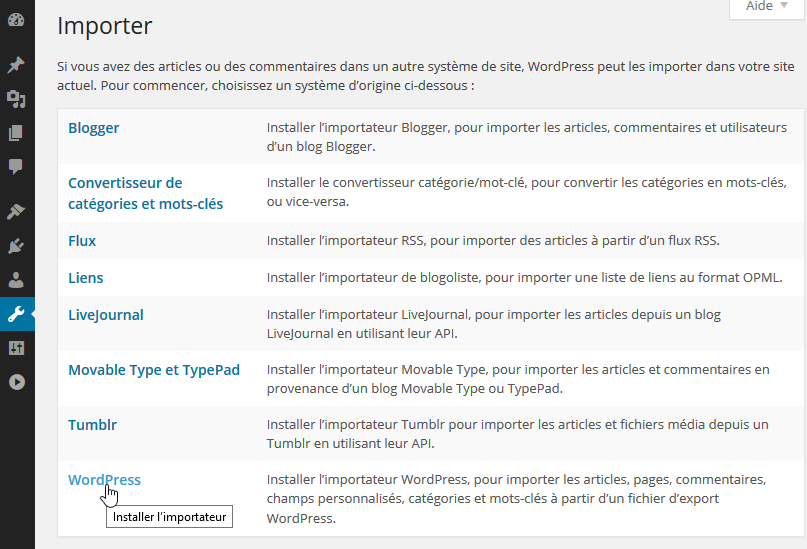
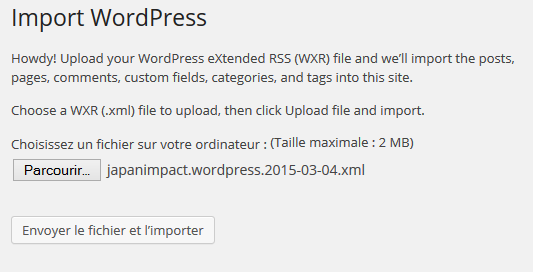
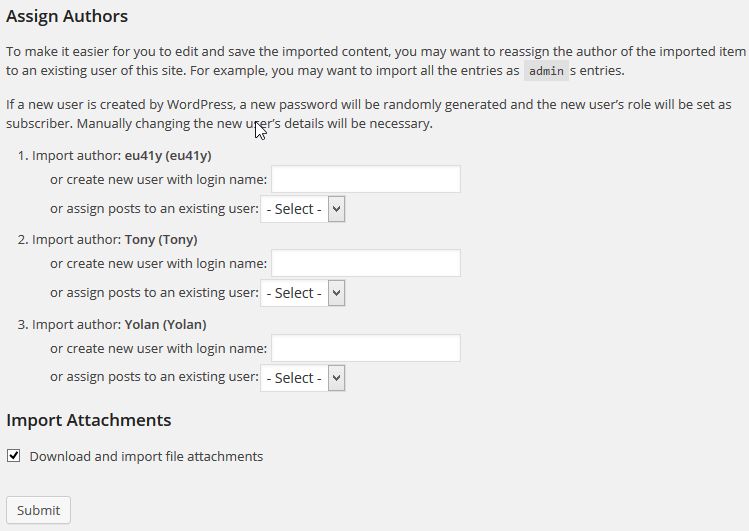
1. J’ai créé un dossier Japan Impact dans le dossier htdocs de XAMPP dans lequel j’ai importé WordPress.  
   
2. Avant d’installer WordPress, il faut créer la base de données dans phpMyAdmin. Pour se faire, je suis allé à l’adresse localhost/ phpMyAdmin et dans l’onglet « Bases de données » j’ai créé « dbJapanImpact ».  
   
3. Pour lancer l’installation, il faut passer par le navigateur à l’adresse : localhost/JapanImpact/wp-admin/install.php
4. Je suis rapidement arrivé sur une page de configuration pour la base de données.  
   
5. Ensuite, je suis arrivé sur la page de configuration du site  
   

## Importation des données fournies par le client

### Thème

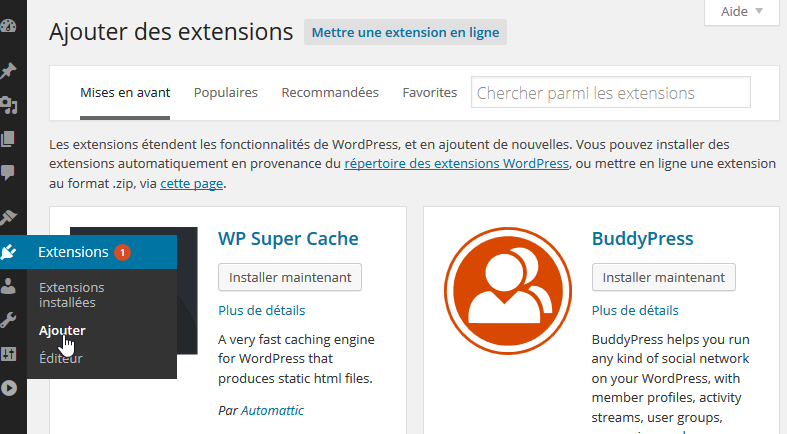
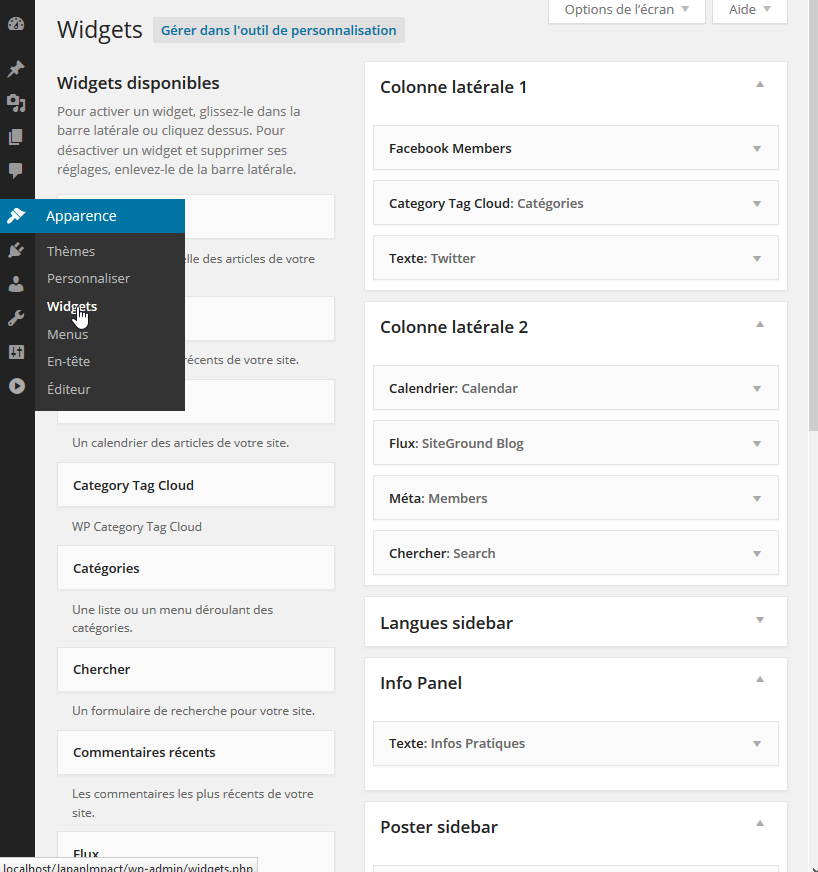
1. J’ai copié le dossier du thème dans le dossier wp-content/themes/ du site.
2. Maintenant je dois l’activer. Je vais dans la partie administration du site. Je vais donc dans la partie thème de la gestion d’apparence et activer le thème.  
     
   

### Pages et articles

1. Dans les outils, j’ai choisis le sous-menu « Importer ». À la fin de la page, il nous propose d’installer un importateur que j’installe.  
   
2. J’active l’extension et importe mon fichier XML.  
   
3. Je peux choisir à qui assigner les articles écrits. De base, il recrée les auteurs et leur attributs leurs articles respectifs, je ne touche donc à rien et je coche l’importation des fichiers attachés.  
   

### Widget

Les widgets ne sont pas compris dans les données du fichier XML, je dois donc accéder au site d’origine et réinstaller et mettre en place tous les plugins manuellement.

1. J’ai dû installer les plugins « Facebook Members » et « WP Category Tag Cloud. Pour se faire, je suis allé ajouter des extensions.  
   
2. La mise en place des widgets se fait dans la section apparence.  
   

# Création du plugin

## Préparation

Les fichiers du plugin sont à placer dans le dossier /wp-content/plugins/. Je crée donc un dossier « BasicTicketting » (le nom de mon plugin).

Je vais mettre en place une structure de dossiers avec le dossier « class » pour mes classes, « views » pour les parties d’affichages, « images » pour les images à intégrer dans les pages du plugin.

Le dossier « Images » contiendra un dossier par événement.

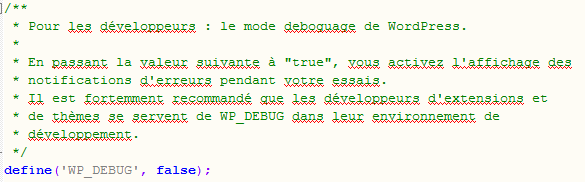
## Réalisation

### Debugging

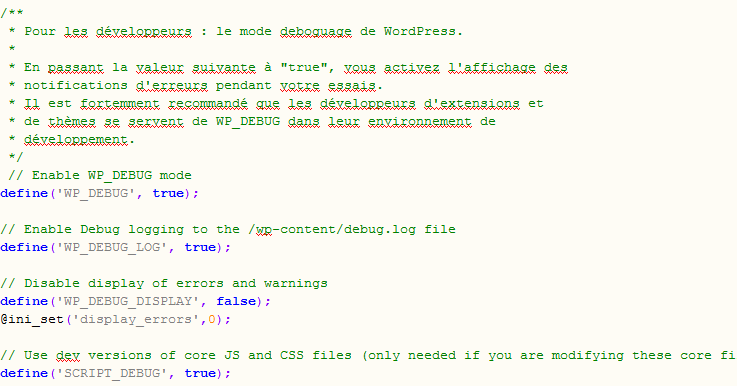
Je commence par activer le debugging dans WordPress. Pour cela je dois modifier le fichier wp-config.php comme indiqué sur la page web :

<https://codex.wordpress.org/Debugging_in_WordPress>

A l’origine, il est désactivé :

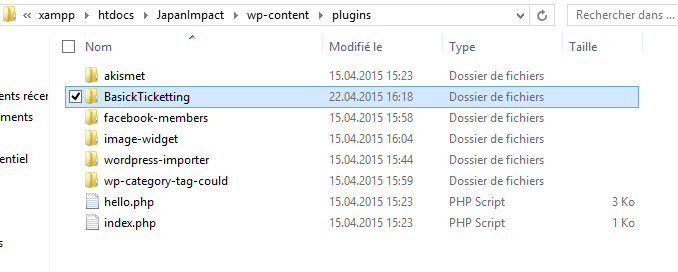


Pour le mettre en place, je le change à true et j’ajoute ces lignes :

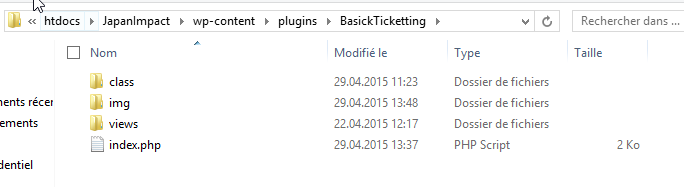


### Base du plugin

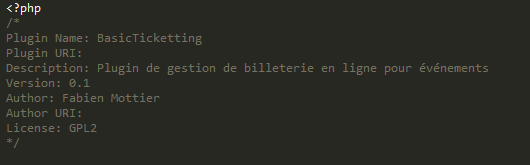
Pour créer un plugin WordPress, je me suis basé sur le chapitre d’OpenClassroom : <http://openclassrooms.com/courses/propulsez-votre-site-avec-wordpress/creer-des-plugins>. Il faut créer un dossier du nom du plugin dans le dossier wp-content :



Dans un fichier de base que j’ai appelé index.php, je vais instancier la classe de mon plugin :



Pour ce faire il faut mettre le nom du plugin en commentaire, d’autres informations importantes sont à noter, mais ne sont pas aussi vital.



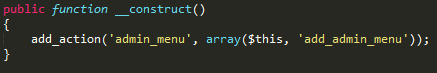
Il faut aussi créer la classe du plugin. Il est important de mettre un préfixe aux noms des classes, pour éviter les doublons avec d’autres plugins. Tout à la fin il faut lancer le plugin.

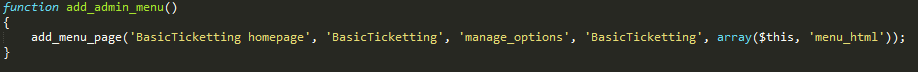
### Base menu backend

J’ai ensuite créé un menu de base avec une page d’affichage. J’ai utilisé ce chapitre OpenClassroom pour se faire :

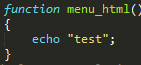
<http://openclassrooms.com/courses/propulsez-votre-site-avec-wordpress/l-administration-2>



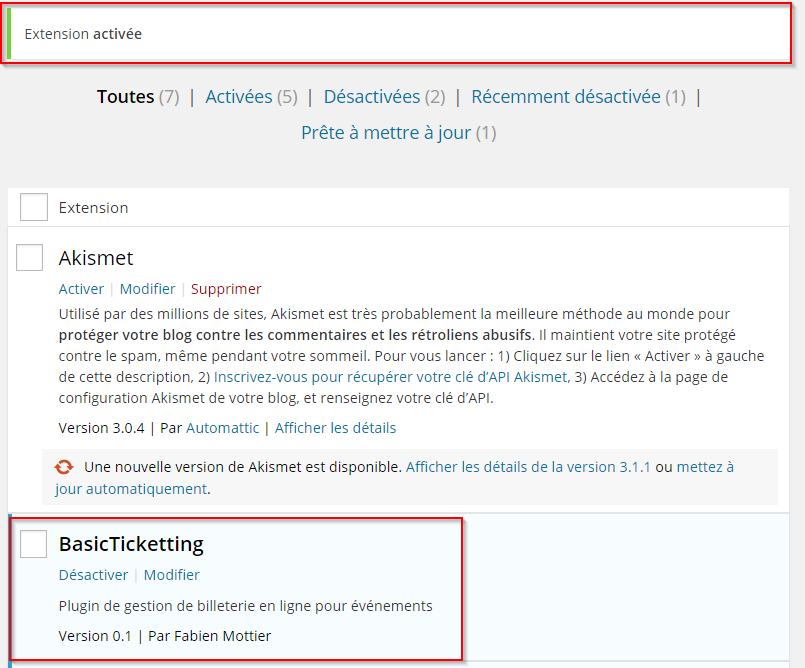
Cette fonction ajoute une action à la création du menu administrateur. Elle crée un menu par la fonction add\_admin\_menu(). A l’intérieur de celle-ci, je peux lancer les fonctions pour créer mon menu et mes sous-menus. Pour le moment je ne crée qu’un menu avec la fonction add\_menu\_page().



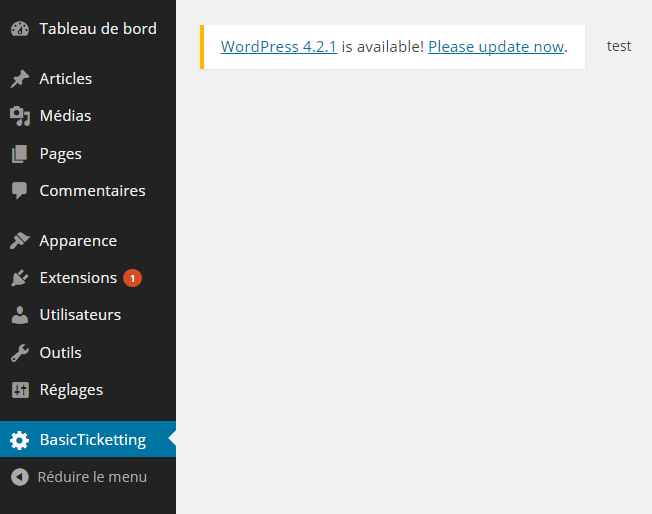
La fonction menu\_html() sert à instancier les éléments à afficher. Pour le moment je crée une simple fonction, mais, plus tard, ce sera une vue de mon dossier view.



Maintenant je vais tester mon code. Dans la partie d’administration de WordPress, dans les extensions, mon plugin est visible et je n’ai plus qu’à l’activer.



Dans la partie supérieure de la page, on nous indique l’extension est activée. Si une erreur se produit, elle s’écrira ici. En bas, on peut voir mon plugin qui a été activé.



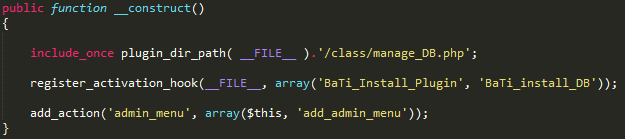
On peut voir mon menu sur la gauche. On peut voir une sorte de notification de WordPress pour effectuer une mise à jour, mais je ne le fait pas pour garder la même version que mon client. Il ne faut pas prendre en compte cette notification. On peut aussi voir le mot « test » que j’ai écrit dans ma fonction menu\_html().

### Création des tables

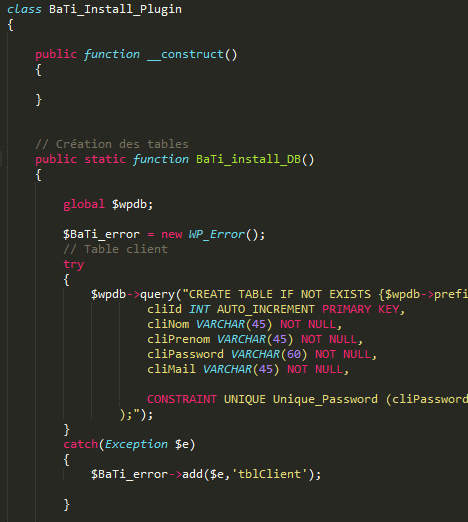
Maintenant je passe à la création de ma base de données. J’ai utilisé le chapitre correspondant d’OpenClassroom :

<http://openclassrooms.com/courses/propulsez-votre-site-avec-wordpress/modifier-la-base-de-donnees>

Pour se faire je vais créer une nouvelle classe pour gérer ma base de données. La fonction pour créer la base de données ne sera à lancer qu’à l’activation du plugin, je dois donc utiliser une fonction spécifique à WordPress register\_activation\_hook(). Il faut lui mettre en paramètre la classe et la fonction voulue. Mais il faut d’abord inclure le fichier de la classe. Mon constructeur est donc modifié ainsi :



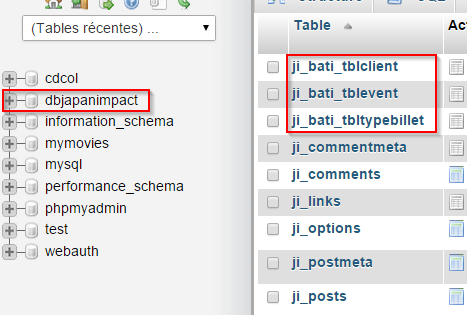
Ma classe de gestion de base de données sera écrite ainsi :



Dans ma fonction d’installation, il y a la déclaration de la variable global $wpdb. C’est une variable WordPress pour gérer la base de données.

La seconde variable $BaTi\_error est là pour la gestion des erreurs, mais je reviendrai plus tard dessus. Le reste de la fonction est une suite de requêtes pour construire la base de données. Pour les noms de table il est aussi utile de rajouter un préfixe pour éviter les doublons.

Quand j’ai essayé de lancer le plugin, Certaines tables ne se sont pas créées :



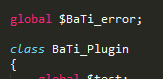
J’avais donc besoin d’une gestion des erreurs. J’ai mis donc ces requêtes dans des try catch. La fonction register\_activation\_hook() ne permet pas d’afficher, ni de retourner une valeur, ni de rajouter des paramètres.

WordPress fournit la classe WP\_Error pour gérer les erreurs, j’ai consulté le codex pour celle-ci :

<https://codex.wordpress.org/Class_Reference/WP_Error>

J’ai donc crée une instance de cette classe avec la variables $BaTi\_error à laquelle j’ajoute une erreur dans chaque catch. Cependant il faut que je déclare cette variable en global pour pouvoir récupérer ces erreurs.

Là j’ai un problème, car on m’a indiqué de déclarer cette variable en dehors de la classe avec le mot global devant comme ceci :



Mais ça ne fonctionne pas, il semble que la variable reste locale.

Voici une liste de site que j’ai consulté sans résultat :

* <https://wordpress.org/support/topic/using-global-variables-in-a-plugin>
* <http://stackoverflow.com/questions/12638734/how-do-i-declare-a-global-variable-in-php-i-can-use-across-templates>
* <http://wordpress.stackexchange.com/questions/89263/how-to-set-and-use-global-variables-or-why-not-to-use-them-at-all>
* <https://codex.wordpress.org/Global_Variables>
* <http://betterwp.net/using-global-variables-in-wordpress/>
* <http://code.tutsplus.com/tutorials/a-practical-use-of-wordpress-global-variables--cms-20854>
* <https://gist.github.com/aahan/7444046>
* <http://pressupinc.com/blog/2013/09/global-variables-scope-php-wordpress/>
* <http://tweetpressfr.github.io/blog/dev/variables-globales-wordpress.html>
* <http://www.wordpress-fr.net/support/sujet-38830-resolu-variables-globales>
* <http://comogene.com/wordpress/themes-tips-wordpress/variable-globale-page-theme-wordpress/>
* <http://weblogtoolscollection.com/archives/2007/06/06/global-variables-and-the-wordpress-loop/>

N’ayant pas réussi à mettre en place la gestion d’erreurs pour la fonction de création des tables, j’ai enlevé les try catch et je vais y aller par tâtonnement. Comme les deux tables qui n’ont pas marchés sont celles qui ont des clés étrangères, je vais me baser là-dessus pour commencer et utiliser la page :

<http://dev.mysql.com/doc/refman/5.6/en/create-table-foreign-keys.html>

J’ai été regardé dans les logs. Le fichier se trouve au chemin :

N:\xampp\htdocs\JapanImpact\wp-content\debug.log

Il m’indique une erreur mysql 150. Elle est en rapport avec les clés étrangères. J’ai essayé de suivre l’idée de ce site :

<http://www.troyfawkes.com/solved-mysql-error-1005-cant-create-table-errno-150/>

C’est-à-dire de rajouter unsigned aux colonnes, mais cela ne fonctionne pas. Je crée donc une nouvelle discussion sur le forum WordPress du site [www.developpez.net](http://www.developpez.net) .

L’adresse de la discussion est :

<http://www.developpez.net/forums/d1516774/php/scripts/blog/wordpress/creation-plugin-probleme-creation-table/#post8230995>

Je rajouterai dans les annexes un PDF contenant la discussion.

Je vais essayer de suivre les indications sur cette page et de définir tout ce que je peux pour éviter que les valeurs par défaut ne correspondent pas et bloque la création de la clé étrangère. :

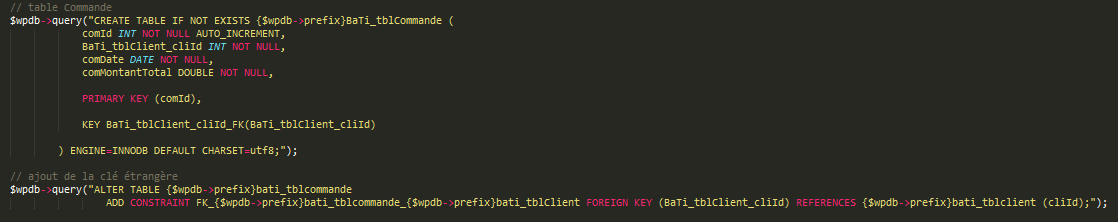
<http://stackoverflow.com/questions/9018584/error-code-1005-cant-create-table-errno-150>

J’ai demandé de l’aide sur le forum de developpez.net :

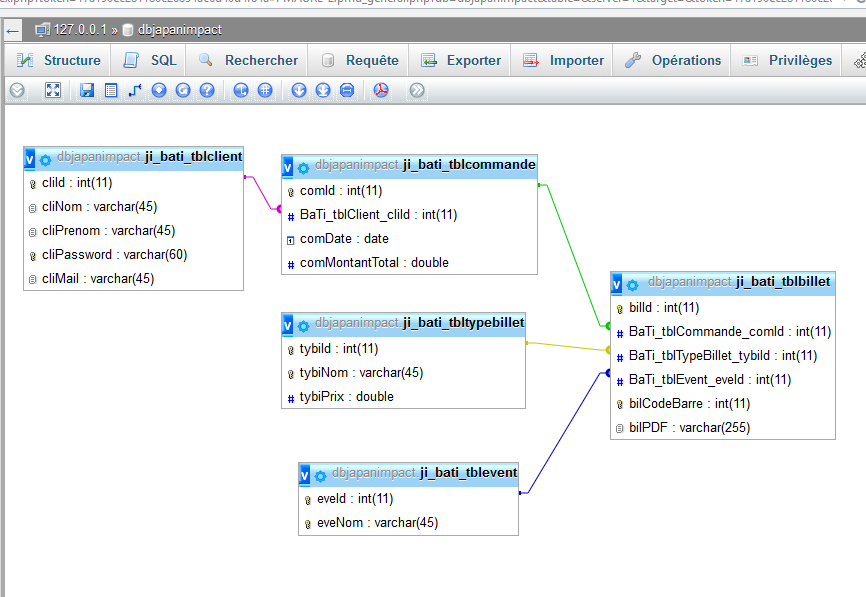
<http://www.developpez.net/forums/d1516774/php/scripts/blog/wordpress/creation-plugin-probleme-creation-table/>

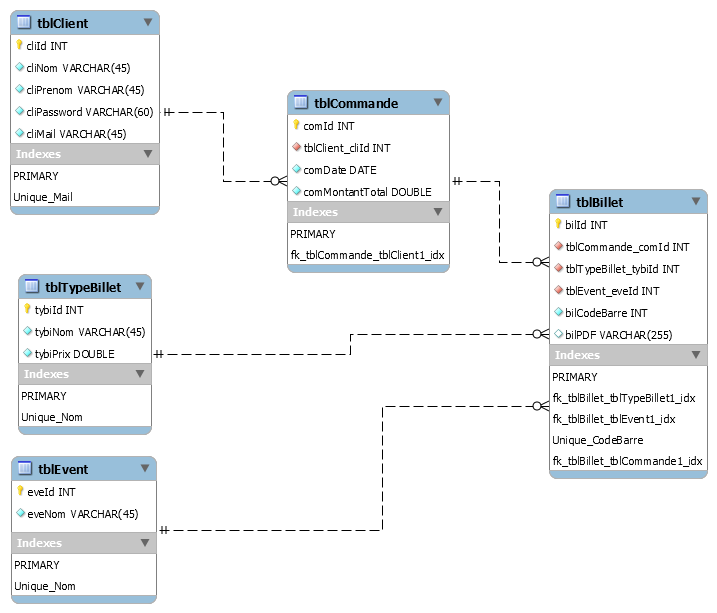
J’ai une copie de la discussion dans les annexes. Une personne m’a proposé d’utiliser un « custom post type », mais je ne l’ai pas utilisé. Cette fonctionnalité permet de créé un élément WordPress qui remplace en quelque sorte une table ce qui serait plus simple à mettre en place et à utiliser, mais cela va bien pour des choses simples. Ici j’utilise des clés étrangères et des indexes uniques, il me faut donc créer des tables.

J’ai trouvé la solution à mon problème de clé étrangère à l’aide de monsieur Thibault Schönnman. Je dois d’abord créer la table avec une clé, puis modifier la table et créer la contrainte par la suite comme ceci :



Pour vérifier le résultat, je suis allé dans la vue concepteur de la base de données dans PHPMyAdmin et j’ai comparé avec mon schéma:



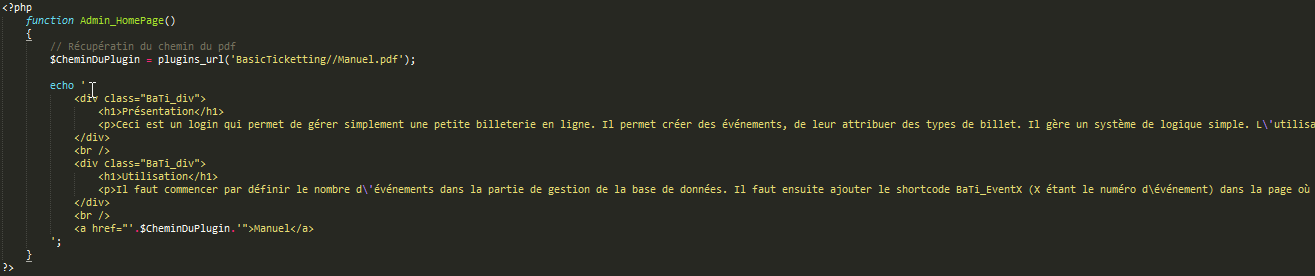


La structure est la même.

### Administration

Je vais à présent mettre en place la partie administrative du plugin qui permettra à l’utilisateur de gérer le plugin. Je vais donc créer la page du menu et des sous-menus.

Je commence avec la home page qui appartient donc au menu. Je peux récupérer la base de menu que j’ai faite précédemment. Il faut donc que je créé une view :



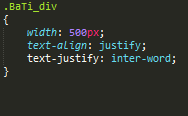
C’est un fichier PHP dans le dossier views. Il faut donc que j’inclue le fichier dans le plugin :



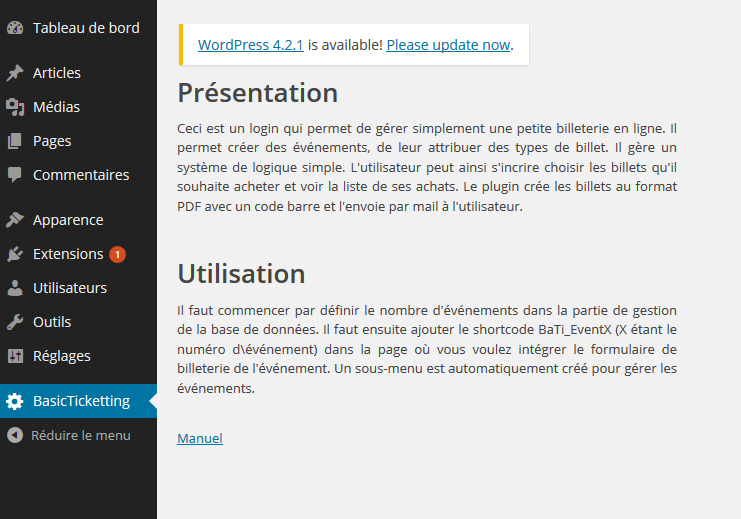
Je sais qu’il existe des classes WordPress utilisables dans le CSS, mais j’ai une préférence de pouvoir le modifier comme je le souhaite et donc de créer un fichier CSS. Pour gérer l’affichage avec un CSS, c’est un peu particulier, car il y a un unique header global pour WordPress. Il faut donc l’ajouter au registre des styles et le mettre à la suite des autres. Il faut donner un nom unique au script.



Script :



Voilà le résultat :



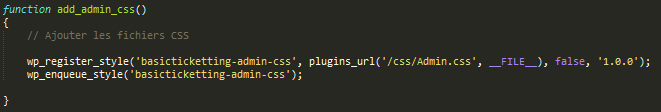
Le client a fait la mise à jour de WordPress, je fais donc pareil et une fois cette mise à jour effectuée j'ai découvert que l’importation du CSS ne fonctionne plus. J’ai été voir dans les logs et ils m’ont affiché ceci.



Il semble que j’essaie d’ajouter au registre mon CSS avant que le registre ne soit créé. J’ai dû donc les ajouter au travers d’une fonction qui se lance lorsque la fonction admin\_enqueue\_scripts().



Par ailleurs le constructeur de la fonction pour rajouter au registre le fichier CSS :



Une chose importante est que chaque fois qu'on modifie le CSS, il faut incrémenter la version (le 1.0.0). Sinon il garde l'ancienne version en mémoire.

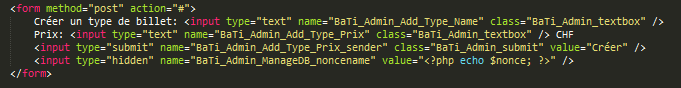
Le lien manuel permettra d’accéder au fichier PDF de documentation.

Je vais maintenant passer à la création du sous-menu de gestion de la base de données grâce à la fonction add\_submenu\_page(). Il faut lui passer comme premier paramètre l’identifiant du menu. On ne peut pas faire un sous-menu d’un sous-menu par contre.

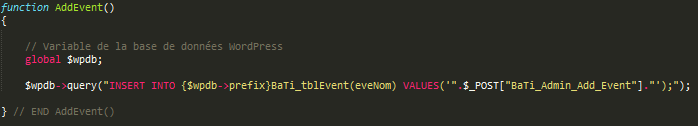


Je reproduis ensuite le même schéma que pour le menu en créant la vue, l’incluant, et ajoutant les CSS nécessaire dans mon fichier Admin.css.

A l’intérieur de la vue je crée deux types de formulaires, deux qui font des insert dans les tables créées :



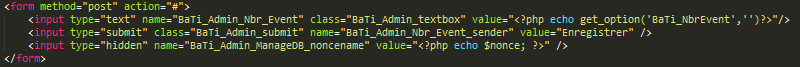
En fonction du formulaire appelé, j’appelle une fonction qui fait un insert :



Le deuxième type de formulaire utilise les options de WordPress. Pour l’écriture de ce formulaire et l’utilisation des options je me suis basé sur la vidéo :

<https://www.youtube.com/watch?v=8ph0iRs_Acc>

Voici le formulaire :



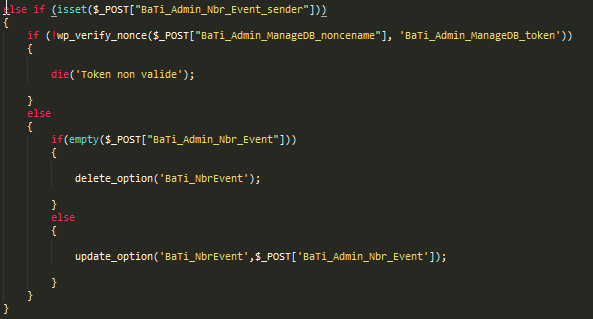
WordPress met en place une table wp-option qui met en place un système de clé-valeur avec le nom de l’option et sa valeur que l’utilisateur entre. L’avantage c’est qu’il existe des fonctions faits pour cette table. Je l’utilise donc pour des valeurs qui ne sont pas utiles à l’archivage.

Dans les deux types de formulaire, j’utilise un input caché. Il sert à sécuriser la mise à jour de valeurs. WordPress crée un token lié à la page appelé nonce :



Si notre token ne correspond pas à celui de la page, on ne peut pas faire la modification, on doit donc ouvrir la page pour effectuer la modification.

La fonction get\_option() vue précédemment ne permet que d’afficher la valeur de l’option. D’abord je contrôle qu’un appel au formulaire avec les options soit effectué, puis si le token est valide. Ensuite je contrôle la valeur entrée par l’utilisateur, s’il est vide je détruis l’option qui ne se fait que si elle existe, sinon je fais un update qui va soit modifier, soit créer l’option.



On peut voir apparaître l’option dans la table wp-option :



Je passe ensuite à la création des sous-menus des événements. Je vais boucler un même sous-menu avec une même view. Dans cette vue je vais devoir récupérer le numéro de l’événement. Lorsque j’accède à la page du sous-menu, WordPress m’envoie automatiquement par la méthode GET le nom du sous-menu :

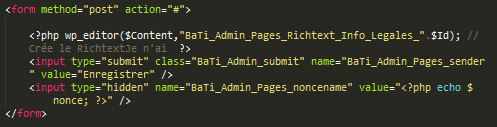


Je n’ai plus qu’à récupérer le nombre à la suite de BaTi\_Event :

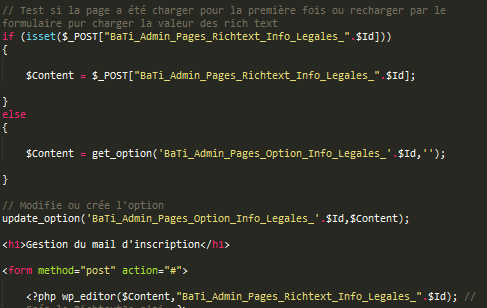


Pour la structure de la gestion des événements, j'ai décidé d'utiliser un seul formulaire, mais réparti en trois groupes d'options: gestion des pages, gestion de l'événement et gestion des mails. Je cacherai les options et afficherai uniquement celles du groupe sélectionner.

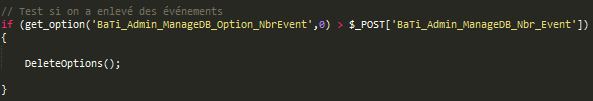
Pour les paragraphes je vais utiliser des rich text grâce à la classe WP\_Editor. Je dois donc créer une variable qui fait le pont entre la valeur du rich text et de l’option. Dans le formulaire je vais simplement ajouter le rich text comme ceci :

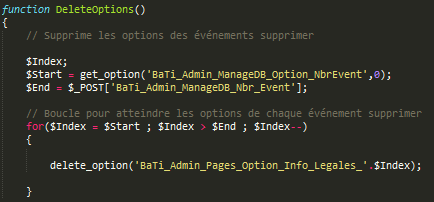


Pour gérer la récupération et affichage de la valeur, je vais d’abord tester si j’envoie le formulaire pour sauvegarder la valeur. Si c’est le cas je rentre dans une variable la valeur provenant de l’envoie de la valeur du rich text créé par WP\_Editor dans le formulaire, sinon je récupère la valeur de l’option par la méthode get\_option qui va soit récupérer la valeur, soit renvoyer la valeur entrée donc une chaîne de texte vide. Ensuite cette valeur est envoyée et à l’option par la méthode update\_option() et au rich text .



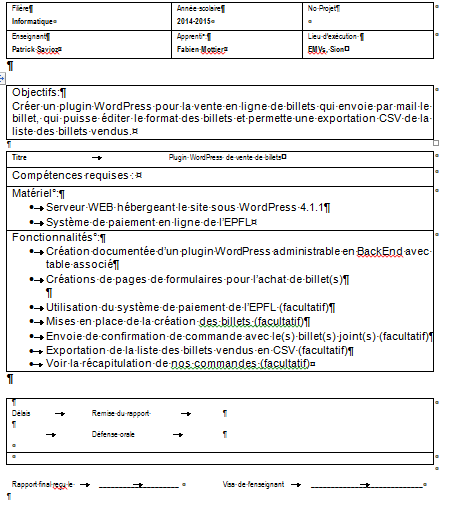
Je dois maintenant faire attention à un détail. Pour éviter de polluer la table des options, je vais supprimer les options des événements supprimés. Donc lors de l’envoie du formulaire du nombre d’événements disponibles, je vais tester si la valeur diminue et si c’est le cas je supprime les options.





Je vais maintenant intégrer les autres options par les mêmes processus pour la partie de gestion des pages de l'événement.

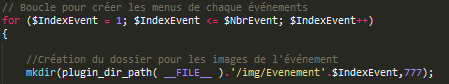
Avec monsieur Savioz, nous avons réévalué le cahier des charges :



Pour la gestion des images, j'ai choisi d'enregistrer les images dans des dossiers spécifiques au plugin pour chaque événement. Je me suis inspiré du tutoriel d'OpenClassroom:

<http://openclassrooms.com/courses/upload-de-fichiers-par-formulaire>

Mais j'ai d'abord dû mettre une fonction pour créer le dossier, j'ai choisi de le faire au même moment que je créer le sous-menu de l'événement comme ceci:



Pour importer les images, j'ai d'abord du intégrer dans le formulaire, la taille maximum d'un fichier. Il faut utiliser un champ caché qui définit le nombre d'octet maximal:



Puis utiliser un input de type file:



Au lieu d'utiliser $\_POST pour récupérer le fichier lors de l'envoie du formulaire, PHP fournit une variable exprès pour, $\_FILES.

Je commence par tester si le transfert à provoquer une erreur:



Lorsqu’on n'envoie aucun fichier le formulaire renvoie la valeur 4, il faut le tester avant d'afficher un message d'erreur:



Si cela passe je test la taille du fichier:



Si cela passe je récupère l'extension du fichier:



Que je compare au tableau d'extension que j'accepte:



Pour tester si une valeur se trouve dans un tableau, on utilise la fonction in\_array():



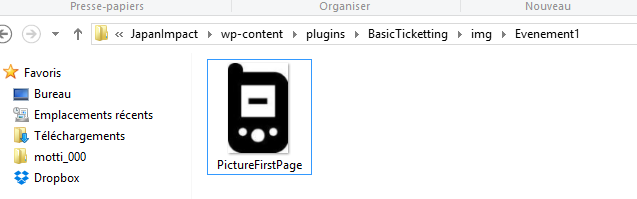
Pour le transfert j'utilise la fonction move\_upload\_file(). Elle demande en premier paramètre de récupérer le fichier temporaire de l'image grâce à la variable $\_FILES["MonImage"]["tmp\_name"] et comme deuxième paramètre le chemin complet de l'image avec son nom. La fonction move\_upload\_file() remplace l'image si elle existe déjà.



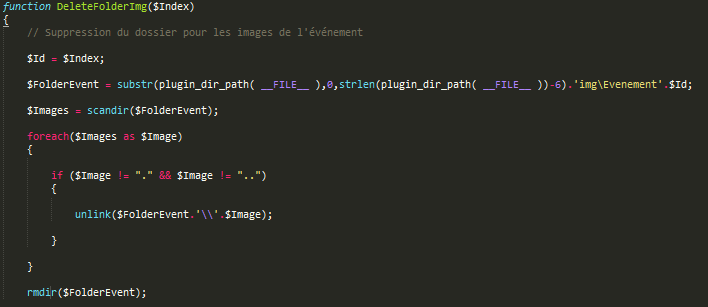
Visuellement dans l'administration on obtient ceci:



Et l'image se trouve au bonne endroit avec le nom qu'on lui a indiqué:



Pour la suppression du dossier d'images à la suppression de l'événement, je l'ai fait en même temps que la suppression des options. Je scanne le dossier, puis je supprime chaque image et ensuite le dossier:



Pour la création d'un combo box je vais utiliser en exemple celui pour choisir le nom d'un événement. J'ai tout d'abord récupéré dans un tableau tous les noms d'événements grâce à la fonction get\_col() qui renvoie un tableau par colonne. En premier paramètre on passe la requête SQL et en deuxième par paramètre le numéro de la colonne. Par défaut, il est à 0, donc si on ne rentre rien, cela nous retourne la première colonne:



Je crée ensuite un select et je boucle les options du select dans un foreach pour chaque nom d'événement. Je teste par ailleurs, si soit la valeur du formulaire, soit la valeur de l'option WordPress vaut le nom d'événement. Si c'est le cas je sélectionne l'option du select.

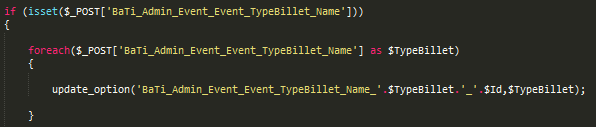


Je voulais d'abord créer des combo box pour le choix des types de billets qui apparaissent dès qu'on sélectionne un, mais finalement j'ai opté pour une liste de check box, nettement plus simples à coder et normalement il n'y aura pas trop de types de billet.

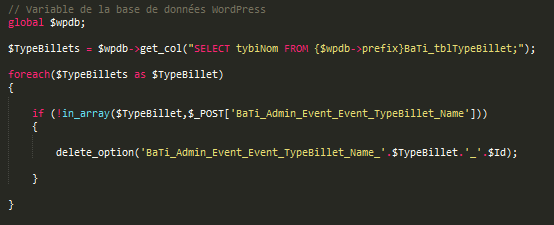
Je réutilise une requête avec la méthode get\_col comme pour le select. et j'utilise un input de type check box:



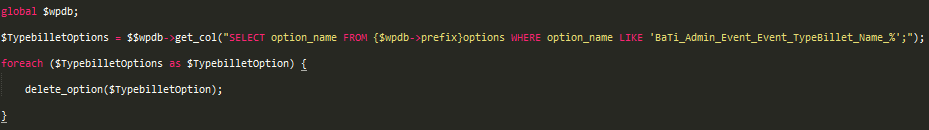
On peut noter que j'utilise un tableau comme nom, ainsi quand j'envoie le formulaire et qu'il y a plusieurs check box d'utilisés, ils se trouvent dans un tableau dans lequel je boucle pour créer mes options:



Pour supprimer les options qui deviennent inutiles, je récupère à nouveau tous les noms de types de billet et je les teste dans ce tableau:



J'ai dû utiliser le nom de l'événement pour le nom de mon option pour avoir une option par type de billet. Je dois donc faire une requête pour récupérer toutes les options commençant par "BaTi\_Admin\_Event\_Event\_TypeBillet\_Name\_":



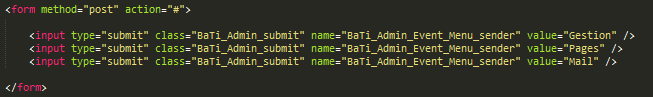
Maintenant je réitère les différentes méthodes précédemment utilisées pour créer le reste de mes options.

Comme j'enregistre une adresse mail, j'utilise la fonction de filtre de PHP pour tester le format de l'adresse mail:



Pour la gestion d'affichage des trois groupes. Je vais créer un formulaire avec trois submit et suivant lequel j'envoie j'affiche le groupe correspondant et envoie en cacher les autres options.

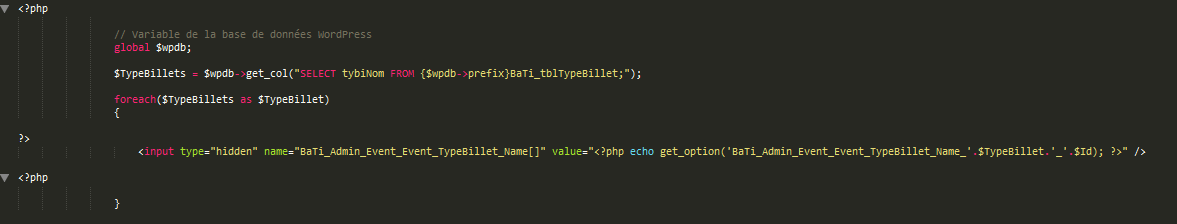
Voici le formulaire:



La valeur du menu choisi est renvoyé dans le formulaire principal en cacher:

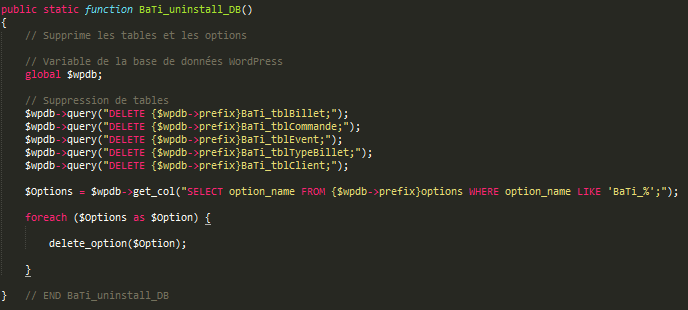


Pour l'envoie caché des types de billets j'ai du à nouveau faire une boucle:



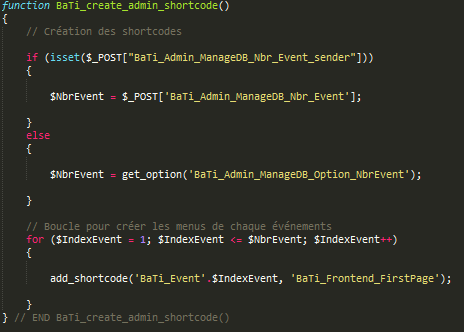
Pour l'ajout de pièce jointe, je voulais au début rajouter dynamiquement des champs, mais à fin d'éviter d'utiliser du JavaScript qui pourrait poser problème à WordPress, j'ai décidé d'enregistrer les pièces jointes dans un dossier de la même manière que j'enregistre les images.

Maintenant que j'ai mis en place l'administration de mon plugin, je crée la fonction qui se lance à la désactivation du plugin et détruit les tables, ainsi que les options:



### Lancement du plugin

Je vais maintenant lancer le plugin sur une page de front end du site. Je dois commencer par créer le shortcode. Le shortcode doit être créé avant l'instanciation du header. Je dois donc créer la fonction du shortcode:

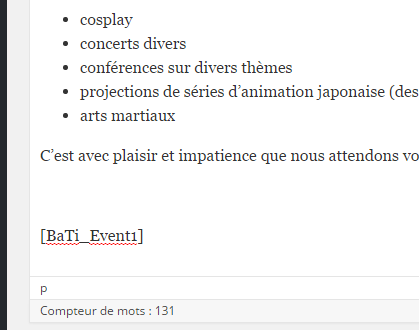


Que je lance à l'initiation au moyen de la fonction spécifique de WordPress:

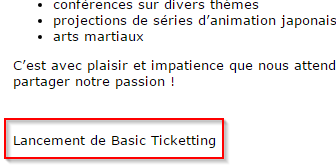


Le paramètre BaTi\_Frontend\_FirstPage est une nouvelle vue créée. Maintenant je n'ai plus qu'à utiliser le shortcode [BaTi\_Event1] dans une page pour lancer l'affichage.

Je l'intègre à la fin de la page de présentation pour effectuer le test:



Résultat:



# Conclusion

Ce fut un projet complexe, car je ne connaissais que très peu le PHP au début du projet et pas du tout la programmation sous WordPress, c’était un projet découverte. Il m’a appris beaucoup de choses, non seulement sur WordPress, mais aussi sur la préparation d’un projet notamment sur une technologie inconnue. Il faut pouvoir analyser la technologie avant de pouvoir planifier quoique ce soit et même dans ce cas, se donner énormément de marge surtout en début de projet. Par ailleurs, il peut être bien de faire une analyse et une révision de la planification après la préparation du projet (création du schéma de base de données, prototype, etc.).

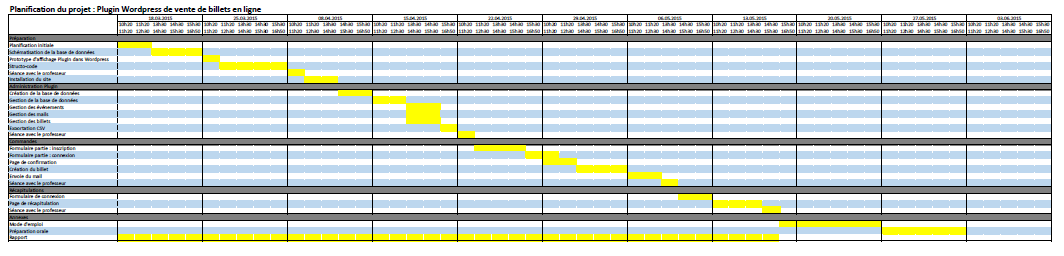
J’ai clairement sous-estimé le côté restrictif de la programmation WordPress. En effet, dans WordPress, tout est lié : il n’y a qu’un seul header, qu’un seul footer, les fichiers CSS et JavaScript sont regroupés dans un registre, certaines fonctions se lancent avant le footer qui est géré par WordPress.

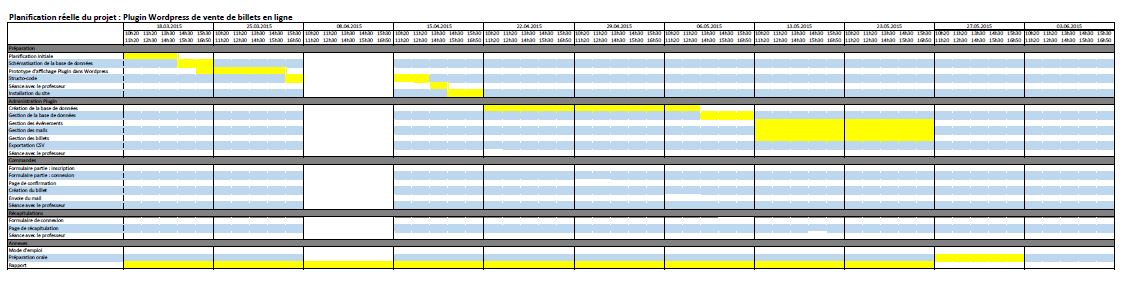
Par ailleurs, la mise à jour de WordPress m’a forcé à reprendre du code, d’où l’importance d’avoir la même version que celle du client pour éviter des problèmes lors du portage.

On peut le voir que j’ai perdu beaucoup de temps au début du projet, puis petit à petit, j’ai compris le fonctionnement et j’avançais plus rapidement.

Je n’ai pas trouvé très utiles les fonctions d’affichage des logs dans WordPress, car la plupart des erreurs bloquaient purement et simplement l’affichage du site. Je devais chaque fois aller regarder dans le fichier de log.

Pour la programmation WordPress, il vaut mieux compartiment le code au mieux, utiliser des variables locales et éviter l’interaction entre elles. On utilise souvent l’envoie de tableau pour transmettre des données. Personnellement j’ai privilégié les formulaires aux variables de sessions et éviter les scripts de JavaScript dans le doute que cela puisse dysfonctionner avec WordPress.





On peut déjà remarquer que je n’avais pas bien vérifié les jours de congé et que rien que cette journée m’a causé une grosse perte. Pour la partie de préparation, j’ai voulu tenter de faire un structo-code, mais connaissant mal la technologie et vu l’ampleur du projet, ce fut une mauvaise idée et j’ai perdu trop de temps avant d’abandonner. Sinon j’aurai tenu les délais. Ensuite on peut remarquer que j’ai mis beaucoup de temps pour la première partie de la création du plugin. C’est celle où j’ai découvert beaucoup de choses et que j’ai pris le plus de retard. Pour la suite de la création de l’administration, il m’a fallu le double du temps prévu.

Il était évident au départ que je ne finirai pas le projet entier, cependant j’ai pu créer le plugin, la première partie. La seconde partie du projet concerne les pages du frontend et la création de la billetterie.

# Annexes

* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_CahierDesCharges\_v1
* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_CahierDesCharges\_v2
* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_JournalDeBord\_v1
* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_PlanificationInitiale\_v1
* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_PlanificationReelle\_v1
* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_SchemaBaseDeDonnees\_v1
* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_PrototypeGeneralPlugin\_v1
* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_PrototypeBillet\_v1
* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_PrototypeAdministrationBackend\_v1
* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_DisscussionClient
* PR-PROJ\_14-15\_Plugin-WordPress-VenteDeBilletEnLigne\_Discussion-Developpez.net
* Code du plugin