Secteur Tertiaire Informatique

Filière « Etude et développement »

Séquence CCP2 _ CMS WordPress

Concevoir un plugin WordPress

Apprentissage

Mise en situation

Evaluation

TABLE DES MATIERES

Version	Date	Auteur(s)	Action(s)

1.0	29/05/2019	Ryad C.	Création du document

Objectifs

Concevoir un module sur un CMS WordPress.

Pré requis

Connaissance en PHP / Orienté Objet en PHP.

Connaissance dans le CMS WORPRESS.

Connaissance en SQL.

Outils de développement

Visual Studio Code

WAMP / XAMPP

Méthodologie

Lecture de cette aide à la conception & des documentations sur WordPress.

Reproduction du code fournit en exemple.

Mode d'emploi

Symboles utilisés :

 Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.

 Propose des exercices ou des mises en situation pratiques.

 Point important qui mérite d'être souligné !

Ressources

Cahier des charges du plugin projet « Vapobar.com »

Lectures conseillées

Codex WordPress.

Plugin HangBook.

FORMATION DWWM – WORDPRESS

CCP 2 – CREEZ UN PLUGIN SOUS WORDPRESS

AIDE A LA CREATION D'UN PLUGIN WORDPRESS

Introduction :

Dans ce TP nous nous proposons de créer un plugin basique pour WordPress.

C'est une première approche pour comprendre les fonctions de bases et la méthode pour créer un module.

Il existe une documentation complète en anglais pour réaliser un plugin :

📖 **Plugin Hangbook** : https://developer.wordpress.org/plugins/

Ce manuel ressence les bonnes pratiques à utiliser pour créer un module.

Réalisez ce TP dans un premier temps puis étudier la documentation pour approfondir le sujet.

Vous devrez rendre la première version du plugin et pour les plus avancés réécrire ce dernier à l'aide d'un ***BoilerPlate***.

Vous fournissez donc aux plus deux plugins au format zip ainsi qu'une documentation technique.

<u>Etape 1 :</u> Structure du plugin

- Créez dans un premier temps un dossier portant le nom « **eudisclaimer** » dans le dossier **wp-content/plugin**.

Le dossier **wp-content** regroupe tous les plugins de votre site.

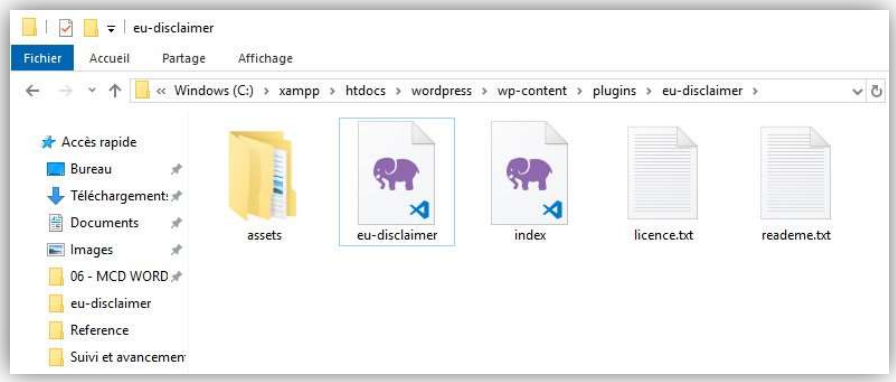
- Créez ensuite un fichier nommé « **eu-disclaimer.php** ».
- Créez un fichier license.txt (informations relatives aux droits d'utilisation de votre plugin).
- Créez un fichier Readme.txt (contient le nom, la description, les instructions d'installation, etc…)
- Créez un dossier « assets » contenant les éventuels fichiers en CSS ou JS.

💣*Les fichiers license.txt et readme.txt sont obligatoires si vous souhaitez publier votre plugin sur le Marketplace de WordPress.*

Enfin, il convient dans chaque dossier de placer un fichier nommé « **index.php »**. En effet, il cache l'arborescence du dossier et masque donc son contenu.

Vous devriez avoir la structure suivante :

Etape 2 : Affichage du plugin dans la partie gestion des extensions de l'administration



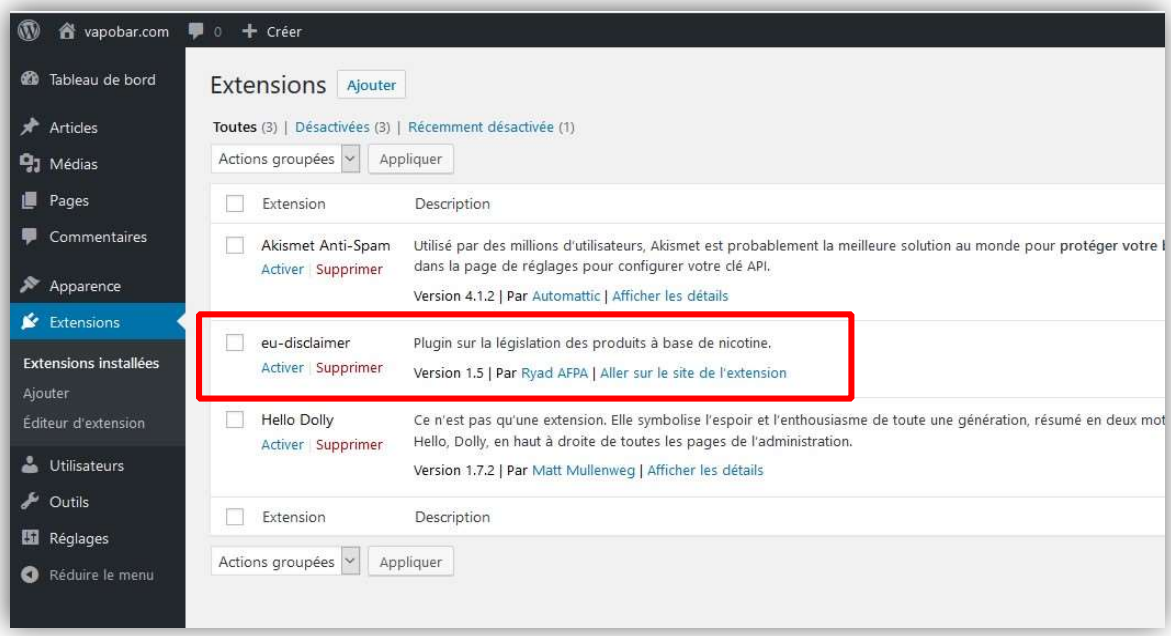
Dans le fichier « **eu-disclaimer.php** » copier / coller le code suivant dans le haut de la page :

```php
<?php
/**
* Plugin Name: eu-disclaimer
* Plugin URI: http://URL_de_l_extension
* Description: Plugin sur la législation des produits à base de nicotine.
* Version: 1.5
* Author: Ryad AFPA
* Author URI: http://www.afpa.fr
* License: (Lien de la licence)
 */
?>
```

L'ajout de ses lignes de commentaires affichera votre module dans le panel administration des extensions.

Toutes ces informations sont obligatoires.

De plus pour diffuser votre plugin dans le Marketplace, vous devez impérativement mettre le lien de votre licence.



A ce stade vous pouvez activer ou désactiver le plugin via le panel des extensions.

Encore une fois c'est l'ajout du commentaire dans votre code en haut de page qui suffit pour que WordPress génère le reste.

Wordpress se charge donc de créer automatiquement le bouton activer / désactiver.

En créant le fichier « **eu-disclaimer.php** », vous avez indirectement créer une extension du fichier « **functions.php** » de WordPress.

En d'autres termes votre fichier communique directement avec Wordpress comme si votre code étant présent dans le fichier « **functions.php** » sans pour autant le modifier directement.

<u>Etape 3 :</u> Affichage du plugin dans le menu de l'administration

Utilisez la fonction **« add_menu_page »** pour afficher un raccourci vers votre plugin.

📖 *Source :*
https://developer.wordpress.org/reference/functions/add_menu_page/

Cette fonction comprend plusieurs paramètres :

- $page = nom de la page.
- $menu = nom affiché dans le menu admin.
- $role = droit offert à l'utilisateur.
- $slug = lien url.
- $fonction = nom de la fonction qui génère le menu.
- $icon : chemin de l'image icône pour personnaliser le menu.
- $position : position dans le menu admin.

```php
//Création de la fonction "Ajouter au menu"
function ajouterAuMenu() {
    $page = "eu-disclaimer";
    $menu = "eu-disclaimer";
    $capacity = "edit_pages";
    $slug = "eu-disclaimer";
    $function = "disclaimerFonction";
    $icon = "";
    $position = 80; // L'entrée dans le menu sera juste en dessous de "Réglages"
    if (is_admin()) {
        add_menu_page($page, $menu, $capacity, $slug, $function, $icon, $position);
    }
}
```

Voici la liste des positions

- 2 : le tableau de bord
- 5 : les articles
- 10 : les médias
- 15 : les liens
- 20 : les pages
- 25 : les commentaires
- 60 : l'apparence
- 65 : les extensions
- 70 : les utilisateurs
- 75 : les outils
- 80 : les réglages

Ensuite nous allons créer un **Hook** !

Mais qu'est-ce qu'un **Hook** ???? Je vous rassure aucun rapport avec le capitaine crochet.

Les **hooks** (ou crochet, hameçon en français) sont des mécanismes permettant d'effectuer une **action supplémentaire** à un instant donné prévu par le système.

En d'autres termes, il s'agit d'une fonctionnalité de **WordPress** permettant l'ajout d'une action supplémentaire au CMS.

📖 ***Hooks*** *: https://developer.wordpress.org/plugins/hooks/*

Il existe deux types de **Hooks** les **Actions** et **Filtres**.

Les **hooks** d'action vous permettent d'ajouter des données ou de modifier le fonctionnement de WordPress.

Créez dans un premier temps une fonction nommé « disclaimerFonction() ».

```php
// hook pour réaliser l'action 'admin_menu'
add_action("admin_menu", "ajouterAuMenu", 10);

//Fonction à appeler lorsque l'on clique sur le menu
function disclaimerFonction() {
    require_once('views/disclaimer-menu.php');
}
```

Cette fonction à pour objectif de charger une page PHP que nous allons créer et ranger dans le dossier **Views** (vues).

Créez donc ce fichier nommé **« disclaimer-menu.php »** puis placez le dans le dossier **Views**.

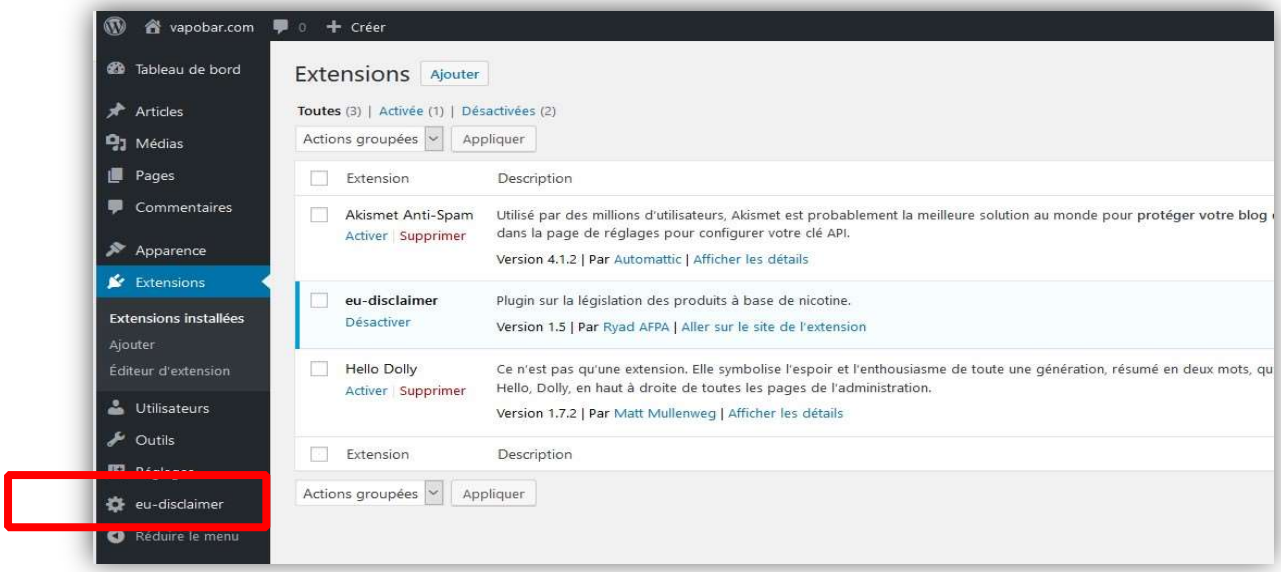
Nous utiliserons l'instruction ***require_once()*** pour appeler le fichier :

📖 **require_once()** : https://www.php.net/manual/fr/function.requireonce.php

Ensuite, il ne vous reste plus qu'à lier cette fonction « disclaimerFonction() » à WordPress avec la fonction **add_action()** qui comme sa traduction en français l'indique va ajouter une action.

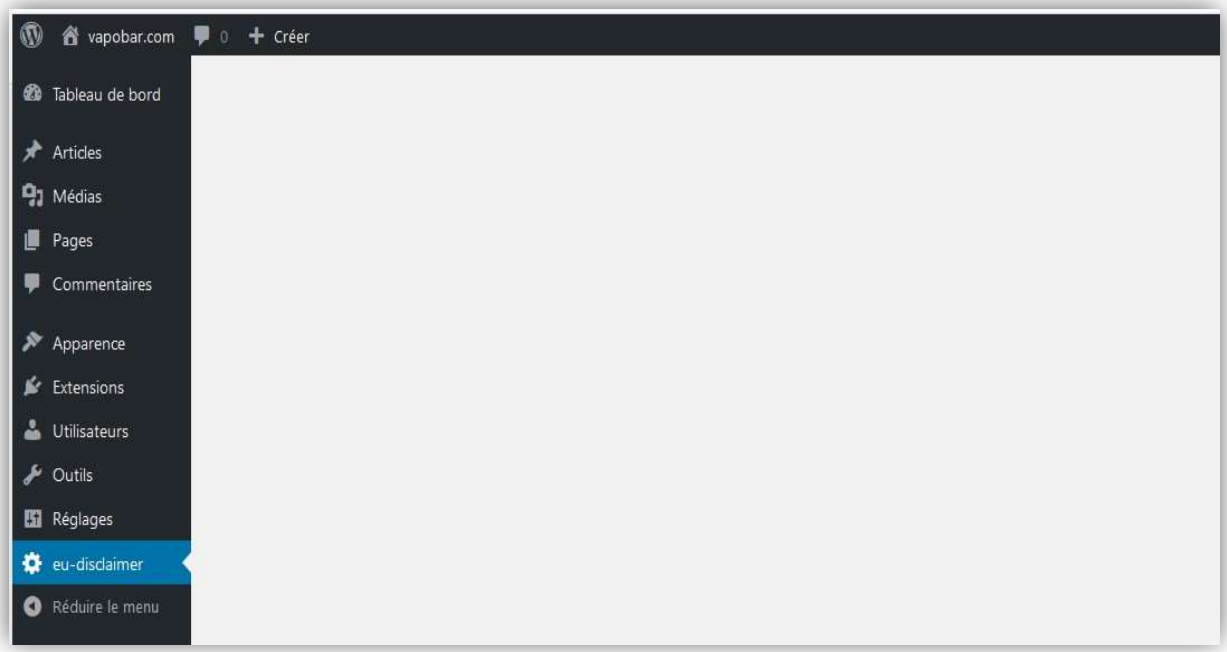
📖 Actions : https://developer.wordpress.org/plugins/hooks/actions/

Voici le rendu après l'ajout du hook d'action :



Un bouton apparait bien à présent dans le menu de l'administration.

Au clic sur le bouton du menu, vous appelez la page « **disclaimer-menu.php** ».



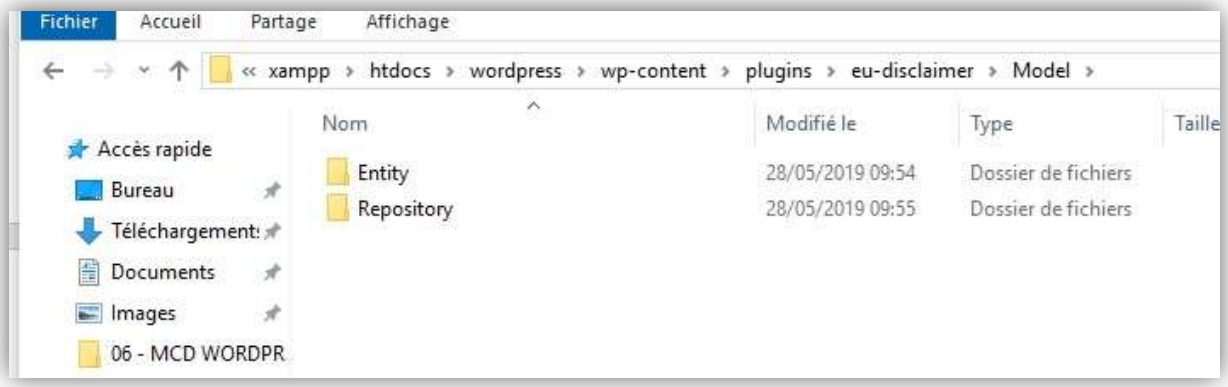
Comme vous le constatez pour le moment, la page est vide !

Nous complèterons cette partie en créant un formulaire.

Mais avant nous allons créer une classe nommée « **DisclaimerOptions »** ainsi qu'une table « **disclaimer_options »** dans la base de données.

<u>Etape 4 :</u> Création de la classe DisclaimerOptions

Vous allez donc créer un dossier **Model** avec un sous-dossier nommé **Entity** et un autre **Repository**.



Dans le dossier **Entity**, nous allons créer une classe nommée **DisclaimerOptions**.

Nous allons créer 3 propriétés et les encapsuler avec des getter et setter (accesseur et mutateur).

```
private $id_disclaimer;
private $message_disclaimer;
private $redirection_ko;
```

$id_disclaimer = identifiant du disclaimer.

$message_disclaimer = contenu du message à afficher.

$redirection_ko = url de redirection en cas de réponse non.

Astuce : Sous VISUAL STUDIO CODE utilisez l'extension « **PHP getters and setters for Visual Studio Code** » pour générer rapidement les getters et setters. https://marketplace.visualstudio.com/items?itemName=phproberto.vscodephp-getters-setters

Créez également le constructeur :

```php
function __construct($id_disclaimer = "Nc", $message_disclaimer = "Nc", $rediction_ko = "Nc" ) {
// function __construct() {
    $this->id_disclaimer = $id_disclaimer;
    $this->message_disclaimer = $message_disclaimer;
    $this->redirection_ko = $rediction_ko;
}
```

Voici le rendu de la classe **DisclaimerOptions :**

```php
<?php
class DisclaimerOptions{

private $id_disclaimer; private
$message_disclaimer; private
$redirection_ko;

function __construct($id_disclaimer = "Nc", $message_disclaimer = "Nc",
$rediction_ko = "Nc" ) {
    // function __construct() {
        $this->id_disclaimer = $id_disclaimer;
        $this->message_disclaimer = $message_disclaimer;
        $this->redirection_ko = $rediction_ko;
    }

/**
* Get the value of id_disclaimer
 */
public function getIdDisclaimer()
{
   return $this->id_disclaimer;
}

/**
* Get the value of message_disclaimer
 */
public function getMessageDisclaimer()
{
   return $this->message_disclaimer;
}
```

```php
/**
* Set the value of message_disclaimer
* @return  self
 */

public function setMessageDisclaimer($message_disclaimer) {
$this->message_disclaimer = $message_disclaimer;
return $this;
}

/**
* Get the value of redirection_ko
 */
public function getRedirectionko()
{
return $this->redirection_ko;
}

/**
* Set the value of redirection_ko
* @return  self
 */
public function setRedirectionko($redirection_ko)
{
$this->redirection_ko = $redirection_ko;
return $this;
}

}
```

Etape 5 : Création de la table disclaimer_options

Rendez-vous dans le dossier **Repository** et créez un fichier contenant une class nommé **DisclaimerGestionTable**.

Dans un premier temps nous allons créer deux fonctions, une pour la création de la table « **disclaimer_options** » et une autre pour la suppression de cette table lors de l'éventuelle désinstallation de ce plugin.

Dans ce fichier, créez deux fonctions PHP.

La première nommée **ceerTable()** :

Nous créons un objet message qui est une instance de la classe « *DisclaimerOptions* ».

Nous alimentons cet objet avec le setter et utilisons la valeur grâce au getter.

Nous utilisons également la variable globale $wpdb.

Cette classe contient de nombreuses fonctions pour interagir avec la base de données de WordPress.

📖 **Wpdb :** https://codex.wordpress.org/Class_Reference/wpdb

```
public function creerTable() {
        // Instanciation de la classe DisclaimerOption
        $message = new DisclaimerOptions();
        // $message = new DisclaimerOptions(0, "Au regard de la loi
européenne,
        // vous devez nous confirmer que vous avez plus
        // de 18 ans pour visiter ce site ?", "https://www.google.com/");

        // On alimente l'objet du message
        $message->setMessage_disclaimer("Au regard de la loi européenne,
        vous devez nous confirmer que vous avez plus
        de 18 ans pour visiter ce site ?");
        $message->setRedirection_ko("https://www.google.com/");
        global $wpdb;
        $tableDisclaimer = $wpdb->prefix.'disclaimer_options';
        if ($wpdb->get_var("SHOW TABLES LIKE $tableDisclaimer") !=
$tableDisclaimer) {
            // La table n'existe pas déjà
            $sql = "CREATE TABLE $tableDisclaimer
```

```
                (id_disclaimer INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY
KEY,
                 message_disclaimer TEXT NOT NULL,
                 redirection_ko TEXT NOT NULL)
              ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;";
            // Message d'erreur
            if (!$wpdb->query($sql)) {
                die("Une erreur est survenue, contactez le développeur du
plugin...");
            }

            // C'est bon
            $wpdb->insert(
                $wpdb->prefix . 'disclaimer_options',
                array(
                    'message_disclaimer' => $message->getMessage_disclaimer(),
                    // 'message_disclaimer' => 'Message01',
                    'redirection_ko' => $message->getRedirection_ko()
                    // 'redirection_ok' => 'URL01'
                ), array('%s', '%s')
            );
            // $wpdb->query($sql);
        }

    }
```

La deuxième nommée **supprimerTable()** :

```
    public function supprimerTable() {
        global $wpdb;
        $table_disclaimer = $wpdb->prefix . "disclaimer_options";
        $sql = "DROP TABLE $table_disclaimer";
        $wpdb->query($sql);
    }
```

La fonction exécute une requête SQL « **DROP TABLE** *NOM DE LA TABLE* ».

Important :

Vous devez importer bien entendu votre classe **DisclaimerOptions** réalisé précédemment au début du fichier « **DisclaimerGestionTable.php** ».

```
//définition du chemin d'accès à la classe DisclaimerOptions
define( 'MY_PLUGIN_PATH', plugin_dir_path( __FILE__ ) );
include( MY_PLUGIN_PATH . '../Entity/DisclaimerOptions.php');
```

Maintenant rendez-vous dans le fichier **« eu-disclaimer.php ».**

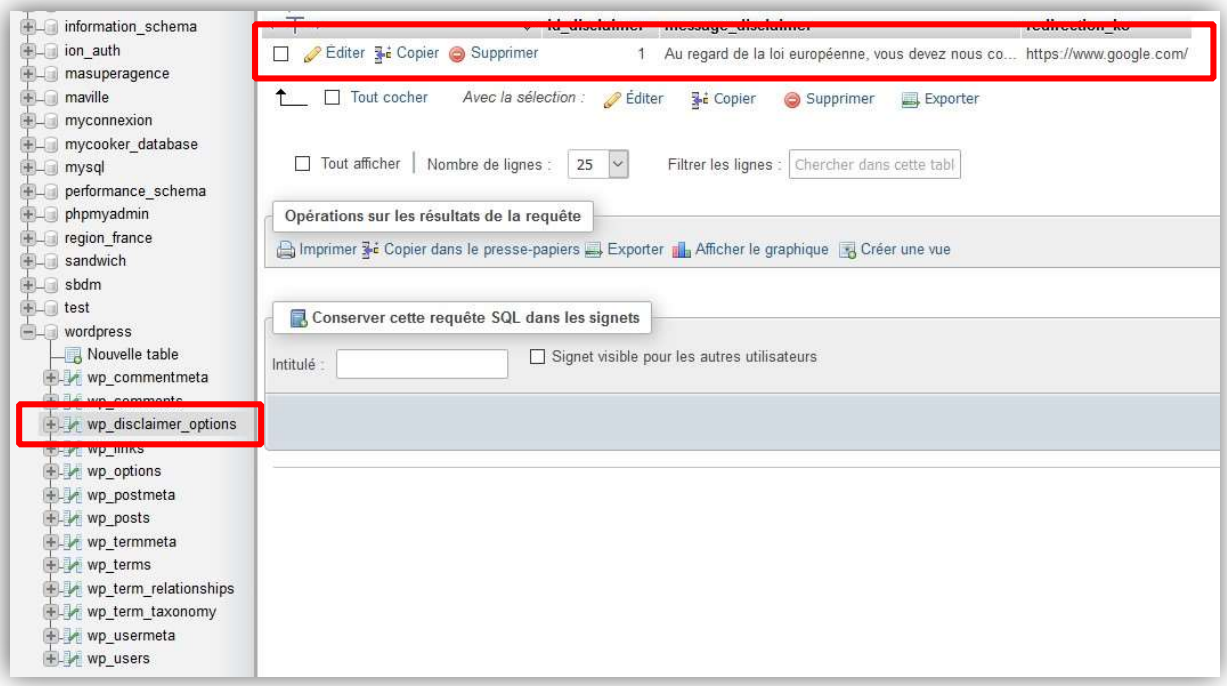
Insérez le code suivant :

```
require_once ('Model/Repository/DisclaimerGestionTable.php');

if (class_exists("DisclaimerGestionTable")) {
    $gerer_table = new DisclaimerGestionTable();
}

if (isset($gerer_table)) {
    // Création de la table en BDD lors de l'activation
    register_activation_hook(__FILE__, array($gerer_table, 'creerTable'));
    // Suppression de la table en BDD lors de la désactivation
    register_deactivation_hook(__FILE__, array($gerer_table,
'supprimerTable'));

}
```

On requiert le fichier **« DisclaimerGestionTable.php »** puis nous créons un objet **$gerer_table**.

On utilise deux hooks, un pour l'installation de la table puis un autre pour la désinstallation de la table.

A ce stade, vous pouvez vérifier le bon fonctionnement de votre code à l'activation du plugin, création de la table et des valeurs par défaut ; puis la suppression de cette même table à la désactivation comme indiqué dans le cahier des charges.



<u>Etape 6 :</u> Création du formulaire du plugin

Rendez-vous dans le dossier **« views ».**

Ouvrez le fichier **« disclaimer-menu.php ».**

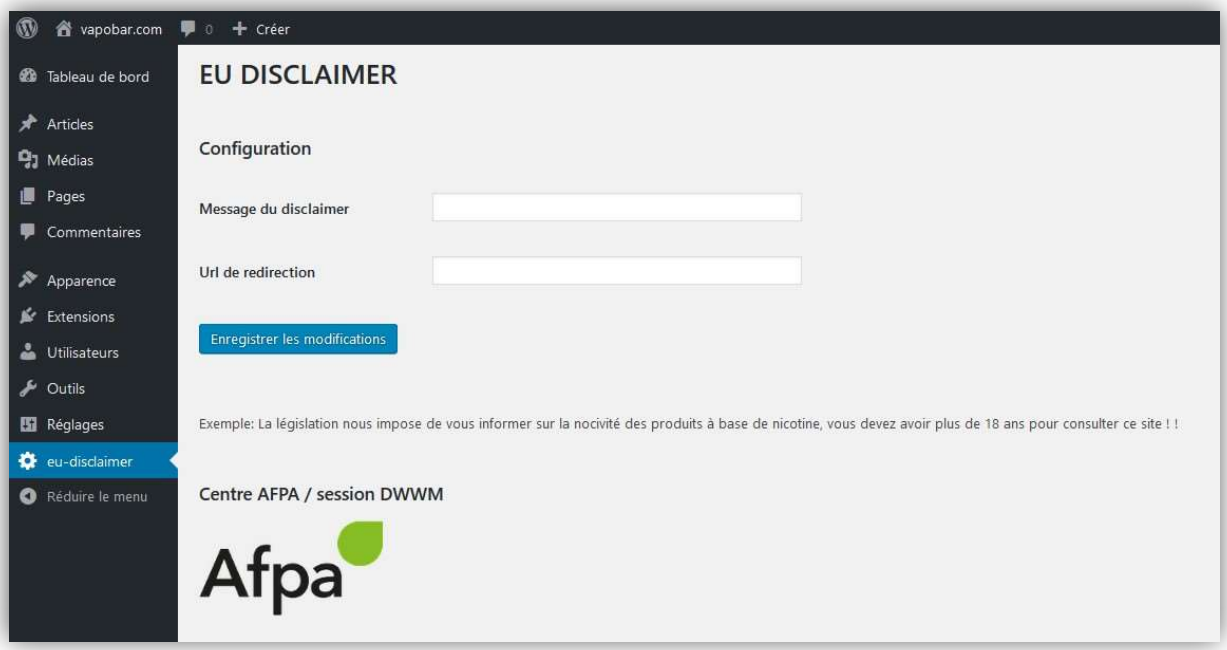
Vous pouvez créer un formulaire avec Bootstrap, dans ce cas vous devez importer les fichiers, placez-les dans le dossier « **assets** » ou vous pouvez utiliser le CDN.

Nous allons simplement reprendre la structure de WordPress pour le formulaire.

Copier le code suivant dans le fichier **« disclaimer-menu.php ».**

```
<h1>EU DISCLAIMER</h1>
<br>
<h2>Configuration</h2>
<p></p>
<form method="post" action="" novalidate="novalidate">
<table class="form-table">
<tr>
<th scope="row"><label for="blogname">Message du disclaimer</label></th>
<td><input name="message_disclaimer" type="text" id="message_disclaimer"
value="" class="regular-text" /></td>
</tr>
<tr>
<th scope="row"><label for="blogname">Url de redirection</label></th>
<td><input name="url_redirection" type="text" id="url_redirection" value=""
class="regular-text" /></td>
</tr>
</table>
<p class="submit"><input type="submit" name="submit" id="submit"
class="button button-primary" value="Enregistrer les modifications"/></p>
</form> <br>
<p>Exemple: La législation nous impose de vous informer sur la nocivité des
produits à base de nicotine, vous devez avoir plus de 18 ans pour consulter
ce site !</p>
<br>
<h3>
Centre AFPA / session DWWM
</h3>
<img src="<?php echo plugin_dir_url( dirname( __FILE__ ) ) .
'assets/img/layout_set_logo.png'; ?>" width="10%">
```

Voici le rendu de la vue :



Nous allons à présent créer la requête pour mettre à jour les données saisies dans notre formulaire.

Dans le dossier « **Repository** », ajouter la fonction PHP suivante :

```php
    static function insererDansTable($contenu, $url) {
        global $wpdb;
        $table_disclaimer = $wpdb->prefix.'disclaimer_options';
        $sql = $wpdb->prepare(
            "
                UPDATE $table_disclaimer
                SET message_disclaimer = '%s', redirection_ko = '%s'
                WHERE id_disclaimer = '%s'
            ", $contenu, $url, 1
        );
        $wpdb->query($sql);
    }
```

La fonction prend deux paramètres (le texte saisie et l'url de redirection).

Ajoutez à présent ce code dans la vue sous le html de votre formulaire :

```php
<?php
```

```
        $text = new DisclaimerOptions();
        $text->setMessage_disclaimer($_POST['message_disclaimer']);
        $text->setRedirection_ko($_POST['url_redirection']);
        DisclaimerGestionTable::insererDansTable($text->getMessage_disclaimer(),
$text->getRedirection_ko());
 ?>
```

Testez le formulaire, vous devriez être en mesure de changer les valeurs et les stockés en base de données.

Remarquez qu'en l'état actuel le fait de cliquer sur le bouton **« envoyer »**, va transmettre des données vides et effacer les données contenues dans la table.

Pour remédier à ce problème, on peut s'assurer que le formulaire ne soit pas vide avant d'être validé.

```
<?php
    if (!empty($_POST['message_disclaimer']) && !empty($_POST['url_redirection'])) {
        $text = new DisclaimerOptions();
        $text->setMessage_disclaimer($_POST['message_disclaimer']);
        $text->setRedirection_ko($_POST['url_redirection']);
        DisclaimerGestionTable::insererDansTable($text->getMessage_disclaimer(),
$text->getRedirection_ko());
    }
?>
```

De plus nous pouvons griser le bouton avec l'aide de JavaScript si les champs sont vides.

Enfin, nous pouvons également afficher le texte déjà présent dans la table avec un **« placeholder »** sur les champs du formulaire.

Dans ce cas nous devons créer une requête pour récupérer les données déjà présente.

Une autre solution consiste à placer les valeurs existantes dans le champ **« value »** des balises du formulaire Html.

Enfin, une autre solution serait d'apporter le champ **« require »** également dans le formulaire.

Autre point, dans un programme il est important de guider l'utilisateur ou de l'informer des opérations réalisées.

Il convient donc d'afficher un message de confirmation, lorsque l'insertion en base de données à correctement été opérée.

Etape 7 : Création du modal d'affichage du plugin

Rendez-vous sur le site de JQuery.

Nous allons intégrer un modal JQuery à afficher sur le front de notre site Vapobar.com

📖 https://jqueryui.com/dialog/#modal-confirmation

Ou

📖 https://jquerymodal.com/

Là encore deux approches sont possibles, soit vous utilisez le CDN mais cela impose d'être connecté sur le web en permanence pour le bon fonctionnement de votre plugin, donc si vous travaillez en local sans connexion internet cela ne fonctionnera pas ou vous pouvez inclure les fichiers dans le dossier **« assets »** que nous vous avons créés précédemment.

Avec le CDN pour le JS :

```
add_action('init', 'inserer_js_dans_footer');

function inserer_js_dans_footer() {
if(!is_admin()):
    wp_register_script( 'jQuery',
'https://cdnjs.cloudflare.com/ajax/libs/jquery/3.0.0/jquery.min.js',
null, null, true );
    wp_enqueue_script('jQuery');
    wp_register_script( 'jQuery_modal',
'https://cdnjs.cloudflare.com/ajax/libs/jquery-
modal/0.9.1/jquery.modal.min.js', null, null, true );
    wp_enqueue_script('jQuery_modal');
endif;
}
```

Avec le CDN pour le CSS :

```
add_action('wp_head','ajouter_css', 1);
function ajouter_css() {
if(!is_admin()):
    wp_register_style( 'modal',
'https://cdnjs.cloudflare.com/ajax/libs/jquery-
modal/0.9.1/jquery.modal.min.css', null, null, false );
    wp_enqueue_style( 'modal' );
endif;
}
```

Vous pouvez à présent faire un clic droit code source de la page sous Firefox et vous assurez que les fichiers soient bien inclus dans la page web du site.

Les fichiers étant bien inclus, nous devons à présent ajouter le contenu du modal.

Prenez à présent l'exemple du site **https://jquerymodal.com/**

Après avoir parcouru les exemples, nous constatez que l'exemple numéro 5 convient à notre projet.

Nous allons créer une fonction pour afficher le modal est son contenu.

Rendez-vous dans le fichier **DisclaimerGestionTable.php**.

Ajoutez la fonction suivante :

```
    static function AfficherDonneModal() {
        global $wpdb;
        $query = "SELECT * from " . $wpdb->prefix."disclaimer_options";
        $row = $wpdb->get_row($query);
        $message_disclaimer = $row->message_disclaimer;
        $lien_redirection = $row->redirection_ko;
        // echo '<div id="monModal" class="modal">
        return '<div id="monModal" class="modal">
        <p>Le vapobar, vous souhaite la bienvenue ! </p>
        <p>'. $message_disclaimer . '</p><a href="' . $lien_redirection . '"
        type="button" class="btn-red">Non</a>
        <a href="#" type="button" rel="modal:close" class="btn-green">Oui</a>
        </div>';
    }
```

La fonction comporte une requête sql qui récupère les données de notre table puis nous affichons ces données dans le modal.

Comme expliquez dans l'exemple 5, nous devons ajouter le code javascript suivant :

```
$("#monModal").modal({
escapeClose: false,
clickClose: false,
showClose: false
});
```

C'est pour nous l'occasion de créer et charger des fichiers dans le dossier des Assets créé précédemment.

Créez donc un fichier nommé **eu-disclaimer.js** dans le dossier **Assets/js** et un autre nommé **eu-disclaimer-css.css** dans **Assets/css**.

Chargeons à présents ces fichiers, retournez dans le fichier **eu-disclaimer.php**.

Ce dernier se comporte comme le fichier contenant toutes les fonctions de Wordpress.

La fonction **plugins_url()** de WordPress permet de charger des fichiers directement dans le dossier.

Ajoutez ces fonctions :

Avec le Assets pour le JS :

```
add_action('init', 'inserer_js_dans_footer');

function inserer_js_dans_footer() {
   if (!is_admin()) :
      wp_register_script('jQuery',
      'https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js',
       null, null, true);
      wp_enqueue_script('jQuery');
      wp_register_script('jQuery_modal',
      'https://cdnjs.cloudflare.com/ajax/libs/jquery-
modal/0.9.1/jquery.modal.min.js',
      null, null, true);
      wp_enqueue_script('jQuery_modal');
      wp_register_script('jQuery_eu',
      plugins_url('assets/js/eu-disclaimer.js', __FILE__),
      array('jquery'), '1.1', true);
      wp_enqueue_script('jQuery_eu');
   endif;
}
```

Avec le Assets pour le CSS :

```
add_action('wp_head', 'ajouter_css',1);

function ajouter_css() {
   if (!is_admin()) :
       wp_register_style('eu-disclaimer-css',
       plugins_url('assets/css/eu-disclaimer-css.css', __FILE__),
       null, null, false);
       wp_enqueue_style('eu-disclaimer-css');
       wp_register_style('modal',
       'https://cdnjs.cloudflare.com/ajax/libs/jquery-
modal/0.9.1/jquery.modal.min.css',
       null, null, false);
       wp_enqueue_style('modal');
   endif;
}
```

Bravo, il ne reste plus qu'à personnaliser le modal grâce aux fichiers que nous venons de créer.

Ajoutez le code suivant dans le fichier eu-disclaimer.js :

```
$("#monModal").modal({
    escapeClose: false,
    clickClose: false,
    showClose: false
});
```

Ajoutez le code suivant dans le fichier eu-disclaimer-css.css :

```
a.btn-red {
    color:#fff;
    background: #FF0066;
    padding: .5rem 1rem;
    display: inline-block;
    border-radius: 4px;
    transition-duration: .25s;
    border: none;
    font-size: 22px;
    font-weight: bold;
}

a.btn-green {
    color:#fff;
    background: #87bb34;
    padding: .5rem 1rem;
    display: inline-block;
    border-radius: 4px;
    transition-duration: .25s;
    border: none;
    font-size: 22px;
    font-weight: bold;
}
```

Il nous reste plus qu'à afficher notre plugin sur la première page du site.

Il existe plusieurs solutions pour visualiser le disclaimer.

<u>Etape 8 :</u> Le shortcode.

```php
add_shortcode('eu-disclaimer', 'afficheModal');

function afficheModal() {
    return DisclaimerGestionTable::AfficherDonneModal();
}
```

Pour afficher le plugin sur la Home page, placez le code PHP suivant, juste en dessous de la balise <body> :

```php
<?php echo do_shortcode('[eu-disclaimer]'); ?>
```

```php
<?php
/**
*     The header for our theme
 *
*     This is the template that displays all of the <head> section and
everything up until <div id="content">
 *
*     @link
https://developer.wordpress.org/themes/basics/templatefiles/#template-
partials
 *
*     @package WordPress
*     @subpackage Twenty_Nineteen
*     @since 1.0.0
 */
?><!doctype html>
<html <?php language_attributes(); ?>>
<head>
    <meta charset="<?php bloginfo( 'charset' ); ?>" />
    <meta name="viewport" content="width=device-width, initialscale=1"
/>
    <link rel="profile" href="https://gmpg.org/xfn/11" />
    <?php wp_head(); ?>
</head>
<body <?php body_class(); ?>>
<?php echo do_shortcode('[eu-disclaimer]'); ?>

<?php wp_body_open(); ?>
<div id="page" class="site">
```

La fonction **« do_shortcode() »** agit comme un raccourci, il doit être placé là ou vous souhaitez que votre plugin apparaisse.

📖 **Shortcode :** https://developer.wordpress.org/plugins/shortcodes/

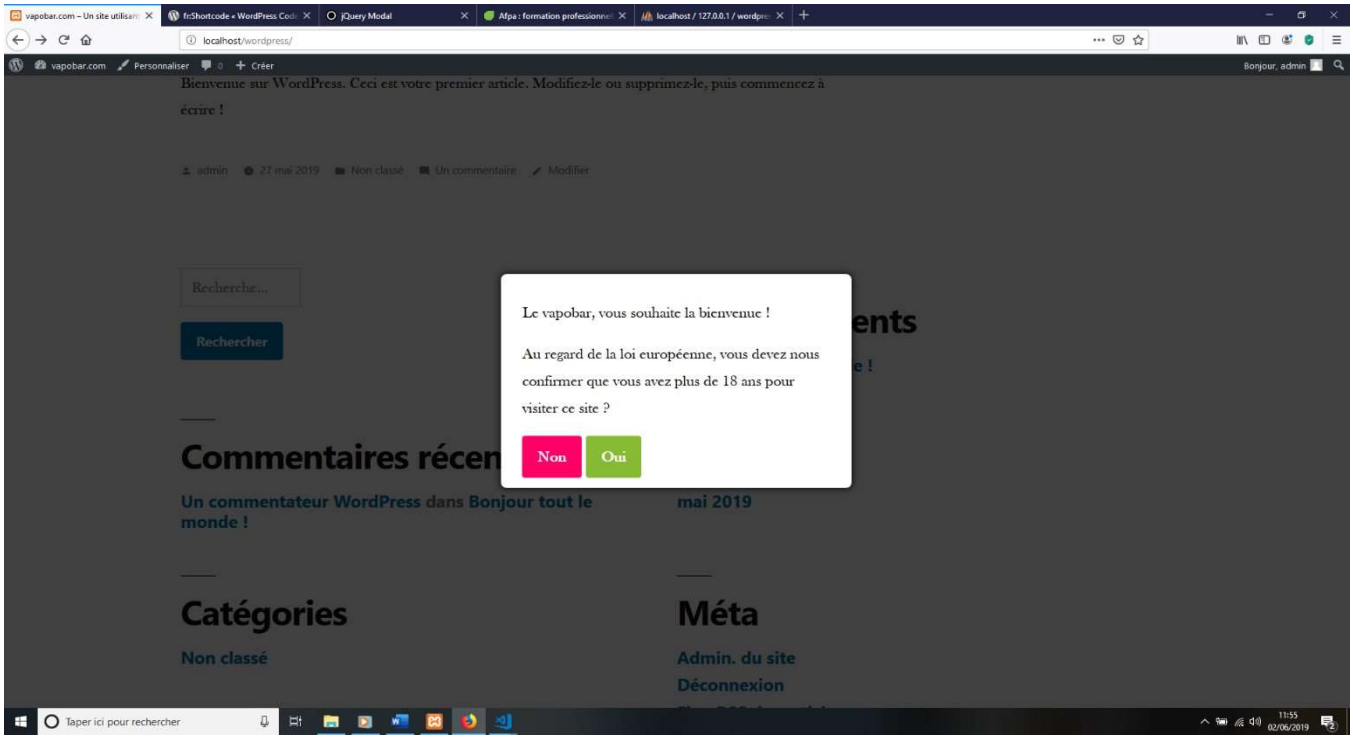
Dans notre exemple nous avons placés le shortcode sous le body, mais vous pouvez le placer librement sous le header, footer, …

💣 ***Utilisez la syntaxe suivante :***

<?php if(shortcode_exists(eu-disclaimer)){ echo do_shortcode('[eu-disclaimer]'); } ?>

La fonction **« shortcode_exists »** évitera d'avoir une erreur dans le cas où le plugin ne serait plus installé.

Au rafraichissement de la page vous devriez voir le modal apparaître.



<u>Remarque :</u> Vous pouvez également créer un article et insérer le ***shortcode*** dans cet article au lieu de le placer dans le code du thème, en utilisant simplement le tag **[eu-disclaimer]**.

L'inconvénient du ***shortcode***, c'est que l'utilisateur du plugin doit ajouter manuellement soit une ligne de code dans le thème soit un insérer le ***shortcode*** dans un article.

L'idéal serait que cette activation se fasse de manière automatique, lors de l'installation du plugin.

Nous allons donc créer un nouveau **hook** pour afficher le modal automatiquement dans le thème sous la balise <body> grâce au hook « wp_body_open ».

Ce hook est présent depuis la version 5.2 de WordPress de Mai 2019.

```
//////////////////////////////////////////
////////////  Affichage     //////////////
////////////  systématique  //////////////
//////////////////////////////////////////
 add_action('wp_body_open', 'afficheModalDansBody');

 function afficheModalDansBody() {
    echo DisclaimerGestionTable::AfficherDonneModal();
}
```

Bien entendu le hook « **add_shortocode** » n'est plus utile et peux être supprimé du plugin.

<u>Etape 9 :</u> Testez votre plugin.

Testez le bon fonctionnement de ce dernier.

Vérifiez le bon fonctionnement de la redirection.

Changez les valeurs dans le formulaire puis testez-le de nouveau.

Bien entendu ce test est sommaire et ne suffit pas.

Vous pouvez consulter la documentation officielle sur les tests unitaires pour les plugins Wordpress pour aller plus loin sur ce sujet.

https://make.wordpress.org/cli/handbook/plugin-unit-tests/

PhpUnit :

https://phpunit.de/

Xdebug :

https://xdebug.org/

WordPress Coding Standards :

https://make.wordpress.org/core/handbook/best-practices/codingstandards/

https://github.com/WordPress-Coding-Standards/WordPress-CodingStandards

Etape 10 : Finalisez le plugin.

- Commentez le plugin.
- Indentez votre code.
- Finalisez le design de ce dernier.

Autre point, nous avons omis de guider l'utilisateur de notre module.

En effet, un bon programme doit être intuitif, et doit informer les utilisateurs lors de certaines actions.

Exemple : *Nous devons ajouter un message de confirmation lors de l'insertion des valeurs dans notre formulaire après l'insertion en base de données.*

Donc nous devons modifier la fonction « **insererDansTable() »** pour fournir un message de confirmation ou d'erreur.

Nous allons ajouter un **try / catch** dans cette fonction :

📖 https://www.php.net/manual/fr/language.exceptions.php

Créez une variable « **$message_inserer_table** ».

Modifiez le code de la façon suivante en ajoutant notre **« try / catch »** pour la gestion des erreurs ainsi qu'un **« return »** du dit message.

```
    function insererDansTable(DisclaimerOptions $otpion){
        $message_inserer_valeur = '';
global $wpdb;               try {
            $table_disclaimer= $wpdb->prefix.'disclaimer_options';
            $sql=$wpdb->prepare(
            "
            UPDATE $table_disclaimer
            SET message_disclaimer = '%s', redirection_ko = '%s'
WHERE id_disclaimer = %s",$otpion->getMessageDisclaimer(),$otpion-
>getRedirectionko(),1
            );
            $wpdb->query($sql);
            return $message_inserer_valeur = '<span
style="color:green; font-size:16px;">Les données ont correctement
été mises à jour !</span>';         } catch (Exception $e) {
            return $message_inserer_valeur = '<span
style="color:red; font-size:16px;">Une erreur est survenue !<span>';
        }
    }
```

Maintenant rendez-vous dans la vue du plugin, ajouter le code suivant :

```
<p><?php if(isset($message)) echo $message; ?></p>
```

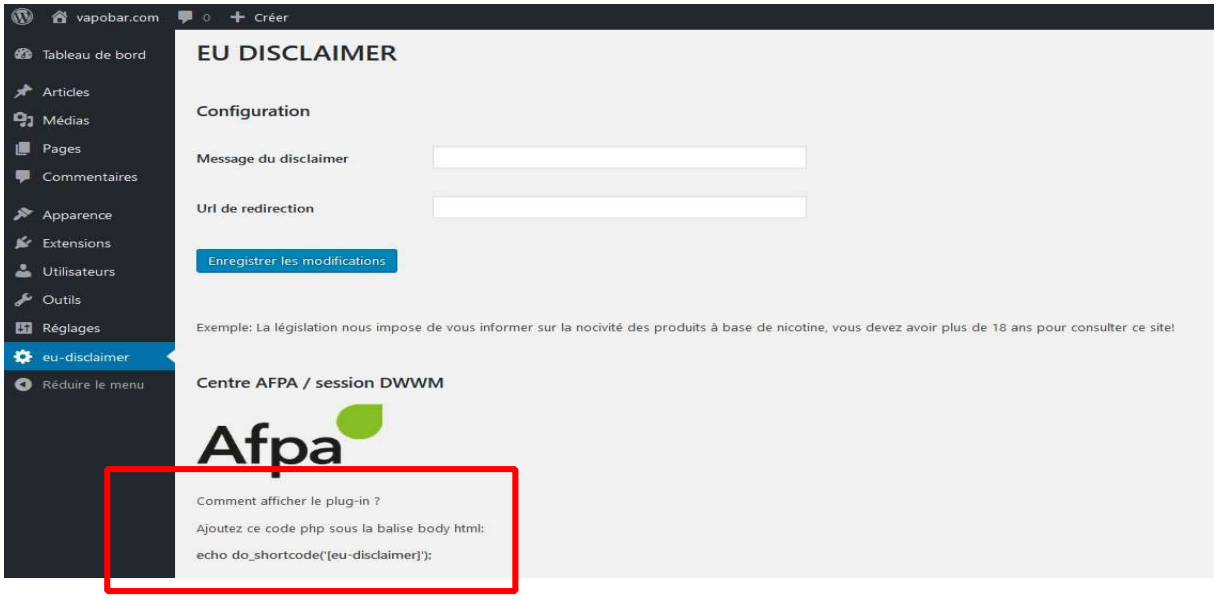
Le reste du code PHP doit être placé en haut de page cette fois ci :

```
<?php
if(!empty($_POST['message_disclaimer']) &&
!empty($_POST['url_redirection'])){
```

```php
    $text = new DisclaimerOptions();
    $text-
>setMessageDisclaimer($_POST['message_disclaimer']);
    $text->setRedirectionko($_POST['url_redirection']);
    $message =
DisclaimerGestionTable::insererDansTable($text);
}
?>
```

Testez à présent en modifiant les valeurs de votre formulaire :

De plus, il serait judicieux d'afficher le « **shortcode** » comme information sur le formulaire du plugin.



Placez également les instructions d'utilisation du « **shortcode** » dans le fichier readme.txt ou dans la description.

Etape 11 : Le Cookie.

Comme vous le constatez, notre plugin fonctionne parfaitement.

Cependant lors de notre phase de test, nous nous sommes aperçus d'un léger problème.

En effet, à chaque fois que nous rafraichissons la page, nous constatons que le plugin est systématiquement affiché.

Or l'idéale, c'est de ne plus afficher ce « **disclaimer »** lorsque que l'utilisateur a déjà cliqué sur oui.

C'est pour cette raison que nous utiliserons un cookie afin de déterminer s'il visite le site pour la première fois sur une période donnée…

Un cookie est donc un petit fichier texte au format alphanumérique déposé sur le disque dur de l'internaute par le serveur du site visité ou par un serveur tiers.

Il peut contenir n'importe quelle valeur, et sa durée de vie est limitée.

Le cookie permet donc de reconnaitre un visiteur lorsqu'il revient sur un site web.

Vous pouvez créer un cookie en PHP ou en JavaScript, la différence entre les deux, est l'endroit où il peut être créé.

Avec PHP, le cookie doit être envoyé avant la balise car il doit faire partie de l'entête de la requête HTTP tandis qu'avec JavaScript, le cookie peut être créé n'importe où dans le document, ce qui est quand même plus pratique !

Nous allons donc travailler principalement dans le fichier **eu-disclaimer.js** présent dans votre dossier **Assets/Js**.

Dans un premier temps nous allons créer une fonction pour créer un cookie :

```
function creerUnCookie(nomCookie, valeurCookie,
dureeJours){
  // Si le nombre de jours est spécifié
  if(dureeJours){
    var date = new Date();
    // Converti le nombre de jours spécifiés en
millisecondes
    date.setTime(date.getTime()+(dureeJours *
24*60*60*1000));
    var expire = "; expire="+date.toGMTString();
    }
    // Si aucune valeur de jour n'est spécifiée
else
      var expire = "";
      document.cookie = nomCookie + "=" + valeurCookie +
expire + "; path=/";      }
```

Ensuite une fonction pour lire le cookie :

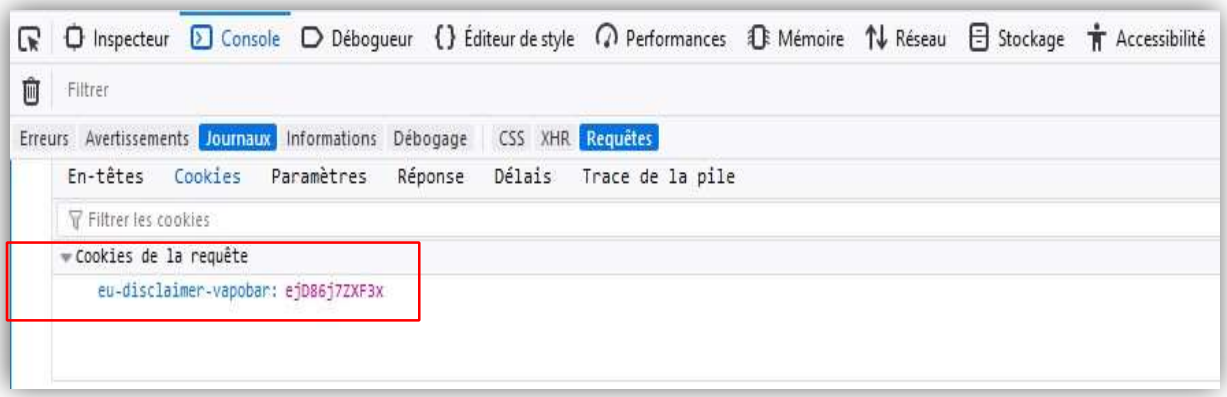
```
function lireUnCookie(nomCookie){
  // Ajoute le signe égale virgule au nom pour la
recherche dans le tableau conteant tous les cookies   var
nomFormate = nomCookie + "=";   // tableau contenant tous
les cookies   var tableauCookies =
document.cookie.split(';');   // Recherche dans le
tableau le cookie en question
  for(var i=0; i < tableauCookies.length; i++) {
var cookieTrouve = tableauCookies[i];
  // Tant que l'on trouve un espace on le supprime
while (cookieTrouve.charAt(0) == ' ') {
    cookieTrouve = cookieTrouve.substring(1, cookieTrouve.length);
    }
  if(cookieTrouve.indexOf(nomFormate) == 0){
return cookieTrouve.substring(nomFormate.length,
cookieTrouve.length);
    }
  }
  // On retourne une valeur null dans le cas où aucun cookie n'est
trouvé
  return null;
}
```

Puis une autre fonction que nous allons lier au bouton « oui » du modal :

```
// Création d'une fonction que l'on va associer au bouton Oui de
notre modal par le biais de onclick. function
accepterLeDisclaimer(){
  creerUnCookie('eu-disclaimer-vapobar', "ejD86j7ZXF3x", 1);
var cookie = lireUnCookie('eu-disclaimer-vapobar');
alert(cookie);
}
```

Au click du bouton la fonction « **accepterLeDisclaimer** » va appeler la fonction « **creerUnCookie** ».

Vous devriez voir apparaitre la valeur du cookie dans la console du navigateur :



Toutefois l'utilisation de la fonction ***onclik()*** de javascript directement dans le code HTML n'est plus recommandé par la fondation Mozilla.

En effet, il est préférable de séparer complétement les différents langages.

Nous allons donc supprimer dans la page HTML au niveau du bouton la balise ***onclick()*** pour la remplacer par du code javascript et plus précisément par la fonction « addEventListener() » (écouteur d'évènement).

📖 https://developer.mozilla.org/fr/docs/Web/API/EventTarget/addEventListener

Rendez vous donc dans le fichier **eu-disclaimer.js** puis au-dessus de la fonction précédemment créée « **accepterLeDisclaimer()** » ajouter la ligne suivante :

```
document.getElementById("actionDisclaimer").addEventListen
er("click", accepterLeDisclaimer);
```

Puis allez dans le fichier ***DisclaimerGestionTable.php*** pour supprimer le *Onclick()* qui se trouve dans la méthode *AfficherModal()* et remplacer le par un Id comme le code suivant :

```
<a href="" type="button" rel="modal:close" class="btn-
green" id="actionDisclaimer" >Oui</a>
```

Maintenant, nous allons placer une condition, si le cookie n'a pas la bonne valeur, alors on affiche le modal.

```
jQuery(document).ready(function($){
if(lireUnCookie('eu-disclaimer-vapobar') !=
"ejD86j7ZXF3x"){
  $("#monModal").modal({
escapeClose: false,
clickClose: false,     showClose:
false
  });
}
});
```

Etape 12 : Sécurité.

Vous pouvez utiliser la fonction PHP **htmlspecialchars()** permet de protéger votre formulaire contre la faille XSS.

- https://www.php.net/manual/fr/function.htmlspecialchars.php

Placez donc cette fonction dans le setter de ***setMessageDisclaimer()*** et sur **set*Redirectionko()*** :

```php
<?php
if(!empty($_POST['message_disclaimer']) &&
!empty($_POST['url_redirection'])){
    $valeur = new DisclaimerOptions();
    $valeur-
>setMessageDisclaimer(htmlspecialchars($_POST['message_disclaimer'
                    ])
);
    $valeur-
>setRedirectionko(htmlspecialchars($_POST['url_redirection']));
    $message =
DisclaimerGestionTable::insererDansTable($valeur);
}
?>
```

La fonction **is_Admin()** de WordPress permet également de sécuriser votre plugin.

Elle restreint ainsi l'accès aux fonctions que vous avez créées.

Seul l'administrateur peut les utiliser. Un tiers ne pourra donc pas les exploiter.

Vous pouvez approfondir le sujet sur le handbook :

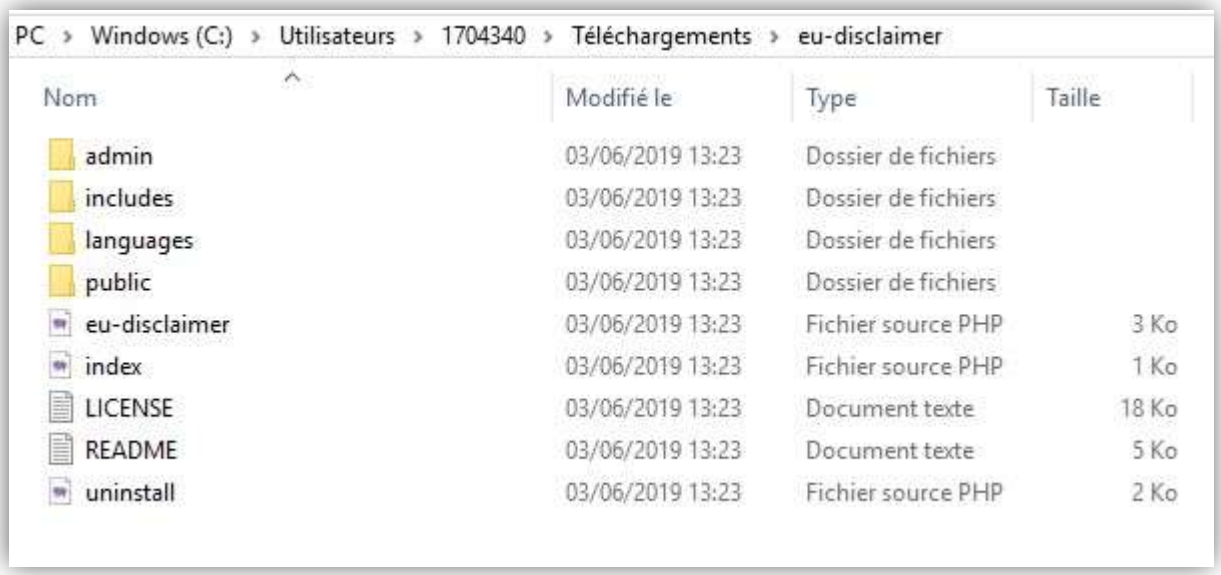
- https://developer.wordpress.org/plugins/security/

Etape 13 : Les boilerplates.

Un boilerplate est une structure de fichier et dossier, conçu pour vous faciliter la tâche dans le développement d'un plugin.

Vous économisez donc du temps car il contient déjà un code prêt à l'emploi.

Architecture du boilerplate :



Les dossiers les plus importants sont :

- **Admin** (concernant la partie administration de WordPress, c'est à dire notre formulaire si on se base sur l'exemple précédent)
- **Public** (partie visible sur le site, le modal dans notre exemple) • **Includes** (contient des fichiers communs)

Le fichier « **eu-disclaimer.php »** a les mêmes fonctions que notre exemple précédent.

Il agit d'une extension du fichier « **fonction.php** » de WordPress.

Il permet donc de lancer le plugin, ajouter des fichiers, déclarer des fonctions etc…

Dans la partie **Includes** vous avez plusieurs fichiers :

- **Activator** : va s'exécuter lorsque vous activez votre plugin dans notre exemple la création de la table.
- **Deactivator** : va s'exécuter lorsque vous supprimer votre plugin dans notre exemple la suppression de la table.
- **I18n** : charger des fichiers de langue en .po et .mo

- Loader : permettre de charger les filtres et les actions (exemples les **hooks** WordPress).
- Le fichier **class-eu-disclaimer.php** permet de déclarer des appels des classes vues précédemment, mais surtout les fonctions d'admin ou de public.

Pour résumé les fichiers les plus utilisés sont :

- **includes/class- eu-disclaimer.php** : pour déclarer les hooks (et éventuellement des custom classes)
- **admin/class- eu-disclaimer -admin.php** : pour écrire ses fonctions destinées à l'admin du site
- **public/class- eu-disclaimer -public.php** : pour écrire ses fonctions destinés à la partie publique du site

Voici un générateur de boilerplate pour WordPress :

https://wppb.me/

Travaux à rendre

Bravo, vous avez réalisés un plugin fonctionnel qui interagit avec la base de données de **WordPress**.

Votre plugin se trouve dans wp-content/plugin.

Zippez votre dossier et testez son installation / désinstallation.

Déposez votre plugin pour évaluation.

Reprenez maintenant votre premier plugin, puis essayez de le réécrire en utilisant le **boilerplate** fournit par le site wppb.me.

Zipper votre dossier et testez son installation / désinstallation.

Déposez votre plugin pour évaluation.

Liens :

Site officielle :

https://fr.wordpress.org/

Télécharger Wordpress :

https://fr.wordpress.org/download/

Codex Wordpress :

https://codex.wordpress.org/fr:Accueil

Code reference :

https://developer.wordpress.org/reference/

Plugin hangbook :

https://developer.wordpress.org/plugins/

Boilerplate Wordpress :

https://wppb.me/

Traducteur :

https://poedit.net/

Voici le lien ou sont regroupés les 54 000 extensions disponibles pour WordPress :

https://wordpress.org/plugins/

WooCommerce :

https://fr.wordpress.org/plugins/woocommerce/

Xdebug :

https://xdebug.org/

WordPress Coding Standards :

https://make.wordpress.org/core/handbook/best-practices/coding-standards/

https://github.com/WordPress-Coding-Standards/WordPress-Coding-Standards

« *Le dossier mu* »

Vous pouvez trouver dans certain cas un dossier mu-plugin qui ne signifie pas « multi-site » mais « must-use ».

Pour plus de renseignements sur mu-plugin, consultez : https://codex.wordpress.org/Must_Use_Plugins.

Remarque : *WordPress MU permet de transformer WordPress en un réseau multi-site, c'est-à-dire que vous pouvez créer plusieurs sites WordPress à partir d'une seule plate-forme principale.*

Lexique & fonction :

Voici les fonctions du sous menu de WordPress.

- **add_dashboard_page() : sous-menu de l'onglet Tableau de bord.**
- **add_posts_page() : sous-menu de l'onglet Articles.**
- **add_media_page() : sous-menu de l'onglet Médias.**

- **add_pages_page() : sous-menu de l'onglet Pages.**
- **add_comments_page() : sous-menu de l'onglet Commentaires.**
- **add_theme_page() : sous-menu de l'onglet Apparence.**
- **add_plugins_page() :sous-menu de l'onglet Extensions.**
- **add_users_page() : sous-menu de l'onglet Utilisateurs.**
- **add_management_page() : sous-menu de l'onglet Outils.**
- **add_options_page() : sous-menu de l'onglet Réglages.**

Fonctions utiles :

- **class_exists() : présence / existence d'un class php**
- **do_shortcode() : exécuter le shortcode**
- **function_exists() : présence / existence d'une fonction php**
- **shortcode_exists() : présence / existence d'un shortcode**
- **add_action() : ajouter une action**
- **add_menu() : ajouter au menu admin**
- **wp_register_style() : enregistrer un fichier de style css .**
- **wp_enqueue_style() : charger le fichier css.**
- **is_admin() : vérifiez si c'est bien l'administarteur.**

Etablissement référent Centre

Afpa de Roubaix.

Equipe de conception

Groupe d'étude de la filière étude – développement.

Collectif de formateurs

Date de mise à jour 08/08/2019