EMO Jeremy

Crit

Documentation

Base de travail codeigniter

Table des matières

[Introduction : 3](#_Toc509494550)

[Liens utiles : 4](#_Toc509494551)

[Ce que vous devez changer : 5](#_Toc509494552)

[Modifications de la configuration de base du framework 6](#_Toc509494553)

[Le fichier config.php : 6](#_Toc509494554)

[Le fichier database.php : 7](#_Toc509494555)

[Le fichier autoload.php : 7](#_Toc509494556)

[Le fichier .htaccess : 7](#_Toc509494557)

[Le fichier index.php : 8](#_Toc509494558)

[Le fichier routes.php : 8](#_Toc509494559)

[Les nouveaux fichiers de configuration : 9](#_Toc509494560)

[Le fichier assets.php : 9](#_Toc509494561)

[Le fichier keys.php : 9](#_Toc509494562)

[Les écrasements de classe de CI : 10](#_Toc509494563)

[MY\_Controller : 10](#_Toc509494564)

[MY\_Model : 11](#_Toc509494565)

[- search() : 11](#_Toc509494566)

[- getById() : 11](#_Toc509494567)

[- create() : 11](#_Toc509494568)

[- myUpdate() : 12](#_Toc509494569)

[- myDelete() : 12](#_Toc509494570)

[- deleteById() : 12](#_Toc509494571)

[Les bibliothèques : 13](#_Toc509494572)

[Template.php : 13](#_Toc509494573)

[MY\_Form\_validation.php : 14](#_Toc509494574)

[Les helpers : 15](#_Toc509494575)

[common\_helper.php : 15](#_Toc509494576)

[session\_helper.php : 15](#_Toc509494577)

[MY\_security\_helper.php : 15](#_Toc509494578)

[MY\_url\_helper : 16](#_Toc509494579)

# Introduction :

Bienvenue sur cette documentation présentant une base de travail pour le framework CodeIgniter 3. J’y présenterai ma configuration de base et tous les ajouts apportés afin de gagner du temps et de regrouper des fonctionnalités communes à de nombreux projets. Une bonne connaissance de PHP orienté objet et du framework sont conseillés, ainsi que du modèle MVC.

En vous souhaitant une bonne lecture,

Jérémy Emo

# Liens utiles :

User Guide de CodeIgniter :

<https://www.codeigniter.com/user_guide/>

Tuto CodeIgniter sur OpenClassrooms :

<https://openclassrooms.com/courses/codeigniter-le-framework-au-service-des-zeros>

# Ce que vous devez changer :

Fichier config.php :

L26 : $config['base\_url'] = 'http://test.vhost';

* Correspond à votre nom de domaine que vous utiliserez pour le site.

Fichier database.php :

L77 : 'dsn' => 'mysql:host=localhost;dbname=test',

* Correspond au DSN de votre base de données

L79 : 'username' => 'root',

* Correspond au nom d’utilisateur pour la BDD

L80 : 'password' => '',

* Correspond au mot de passe de l’utilisateur de la BDD

# Modifications de la configuration de base du framework

## Le fichier config.php :

L38 : $config['index\_page'] = '';

* On supprime le ‘index.php’ de l’url

L380 à 386 :

$config['sess\_driver'] = 'database';

$config['sess\_cookie\_name'] = 'ci\_session';

$config['sess\_expiration'] = 7200;

$config['sess\_save\_path'] = 'ci\_sessions';

$config['sess\_match\_ip'] = TRUE;

$config['sess\_time\_to\_update'] = 300;

$config['sess\_regenerate\_destroy'] = FALSE;

* On change le driver de session pour enregistrer les sessions d’utilisateur dans la base de données. La table utilisée est ci\_sessions, le script pour la créer se trouvant dans le fichier /bdd\_files/init\_sessions.sql. Le cookie contenant les données de session se nomme ci\_session, et pour authentifier la session de l’utilisateur on vérifie son IP, que l’on stocke en BDD afin d’éviter les vols de session et les attaques par cross-site cooking.

L451 : $config['csrf\_protection'] = TRUE;

* On active la protection CSRF, ce qui permet de sécuriser les formulaires créés via la classe form contre les attaques de type Cross-Site Request Forgery.

## Le fichier database.php :

L74 : $query\_builder = TRUE;

* On active l’Active Record, la construction de requêtes SQL par le framework.

L82 : 'dbdriver' => 'pdo',

* On change le driver de BDD du framework par PDO.

## Le fichier autoload.php :

L61 : On charge en automatique les libraries 'session', 'form\_validation', 'template' et 'database'.

L92 : On charge en automatique les helpers 'url', 'security', 'session', 'common' et 'form'.

L106 : On charge en automatique le fichier de config 'keys'.

## Le fichier .htaccess :

On ajoute à la racine du site un fichier .htaccess permettant de réécrire les url de façon propre.

## Le fichier index.php :

L100 : $system\_path = 'framework/3.1.7';

* On change le chemin du dossier system, que l’on a renommé en ‘3.1.7’ et que l’on a stocké dans un nouveau dossier framework. Faire ça permet d’upgrade plus facilement de version de CodeIgniter en cas de mise à jour, et de rollback plus rapidement en cas de problème vu qu’il suffit de changer de chemin.

## Le fichier routes.php :

L52 : $route['default\_controller'] = 'home';

* On change le controller par défaut par Home\_Controller.php.

# Les nouveaux fichiers de configuration :

## Le fichier assets.php :

Il contient tous les noms des liens CSS et des scripts JS chargés sur toutes les pages via la classe Template.

## Le fichier keys.php :

Il va contenir les variables et les clés utilisées sur le site. C’est là que l’on va définir si le site est en maintenance ou non, renseigner ses clés de captcha…

# Les écrasements de classe de CI :

## MY\_Controller :

J’ai ajouté un ensemble de classes enfants utilitaires :

* MY\_Controller extends CI\_Controller :

Permet de récupérer le controller courant.

* Public\_Controller extends MY\_Controller :

Classe pour tous les controllers ne nécessitants pas d’authentification. C’est là qu’est défini le message de maintenance.

* Private\_Controller extends Public\_Controller :

Classe pour tous les controllers nécessitants d’être authentifié.

* Admin\_Controller extends Private\_Controller :

Classe pour les controllers nécessitants d’être un administrateur du site pour y avoir accès.

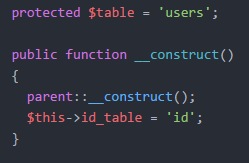
* CLI\_Controller extends CI\_Controller :

Classe pour les controllers ne pouvant être exécutés qu’en ligne de commande. Typiquement des scripts de migration ou des tâches cron.

## MY\_Model :

Chaque classe héritant de MY\_Model doit posséder un attribut $table correspondant à la table dans la BDD qui lui est associée ainsi qu’un champ id par défaut dans le constructeur :

Exemple :



La classe MY\_Model permettant de définir des fonctions par défaut pour chaque classe enfant, plutôt que de définir les mêmes requêtes où seul le nom de la table change.

Les fonctions :

### search() :

Cette fonction permet d’effectuer un SELECT avec en paramètres $select, $filter, $limit et $offset.

$select : le/les champs à récupérer (\* par défaut).

$filter : un array contenant les filtres WHERE. Par exemple :

array('mail', 'a@a.fr') -> correspond à un SELECT \* from xxx WHERE mail = 'a@a.fr'

Dans le cas de plusieurs clauses WHERE :

array( array('mail', 'a@a.fr') , array('nom', 'toto') ) -> correspond à SELECT \* from xxx WHERE mail = 'a@a.fr' AND nom = 'toto'

$limit : la valeur de LIMIT.

$offset : le offset de LIMIT.

### getById() :

Cette fonction permet de rechercher dans une table via un id, qui est le seul paramètre de la fonction.

### create() :

Cette fonction permet d’insérer un array dans la table courante. Exemple :

$data = array (‘nom’ => ‘test’, ‘prenom’ => ‘toto’) ;

create($data) ;

Va insérer dans la table courante test dans le champ nom et toto dans le champ prenom.

### myUpdate() :

Cette fonction permet de mettre à jour la table courante. Le paramètre $filter est le même que pour la fonction search(). Le paramètre $set est un array contenant les données à mettre à jour, de la même forme que la fonction create().

### myDelete() :

Cette fonction permet de supprimer une ligne de la table courante en fonction d’un paramètre $filter (construit de la même manière que les $filter précédents).

### deleteById() :

Cette fonction permet de supprimer la ligne d’id $id dans la table courante.

# Les bibliothèques :

## Template.php :

Il s’agit de la bibliothèque permettant de générer ses pages avec un template. Il est composé des fonctions suivantes :

* render() :

C’est la fonction permettant de générer un page composée de /layout/head.php, la vue à intégrer et de /layout/foot.php. Il est possible de lui passer un objet en second argument pour transmettre des données aux différentes vues composant la page. Dans le cas où la constante ENVIRONMENT est à ‘development’, on active le profiler de CI.

* generateAssets() :

Cette fonction permet de générer toutes les balises <link> et <script> du header paramétrées dans le fichier assets.php.

## MY\_Form\_validation.php :

Cette bibliothèque est une extension de celle déjà existante de CI. J’y ai ajouté les fonctions suivantes :

* validateCaptcha() :

Cette fonction permet la validation du captcha google : invisible Recaptcha ou RecaptchaV2.

# Les helpers :

## common\_helper.php :

Ce helper contient toutes les fonctions utilitaires :

* base64urlEncode() et base64urlDecode() :

Ces deux fonctions permettent l’encodage d’une chaine de caractère en base 64 pour URL. Elles sont pratiques pour utiliser certaines API de google.

* cssLink() et jsScript() :

Ces deux fonctions permettent de générer les balises <link> et <script> à partir d’un nom de fichier ou d’une url. Elles sont utilisées par la bibliothèque Template.php.

## session\_helper.php :

Ce helper contient les fonctions utiles pour le management de sessions :

* setLogin() :

Cette fonction permet de setup les variables de session ‘logged’ et ‘userId’ à partir d’un id utilisateur. Ces variables sont utilisées entre autres pour la gestion d’authentification par les controllers.

* disconnect() :

Cette fonction permet de déconnecter l’utilisateur.

## MY\_security\_helper.php :

Ce helper est une extension du helper security de CI :

* hashPassword() :

Cette fonction permet de hash une chaine de caractère avec l’algorithme par défaut courant de PHP.

* verifyPassword() :

Cette fonction permet de vérifier un mot de passe en fonction d’une adresse mail. Elle permet également de vérifier si l’utilisateur existe.

* generateToken() :

Cette fonction permet de générer un token unique, qui peut être utilisé pour la validation de mail ou encore le reset de mot de passe.

## MY\_url\_helper :

Ce helper est une extension du helper url de CI :

* site\_url() :

Cette fonction est une réécriture de la fonction de base. J’y ai ajouté la possibilité d’y passer en paramètre une url complète. Cela permet de récupérer une url en BDD et de la passer dans la fonction site\_url sans avoir à se soucier si c’est un chemin local ou une URL complète.

* css\_url() et js\_url() :

Ces deux fonctions permettent de construire une url pour les fichiers css et js locaux.

# Les migrations :

Pour migrer simplement une base de données, CI embarque la classe Migration qui permet d’exécuter des requêtes SQL depuis un controller. Les fichiers de migrations sont stockés dans application/migrations. Ceux-ci sont de type « sequential » et doivent donc se nommer sous la forme 001\_nom\_de\_la\_migration.php