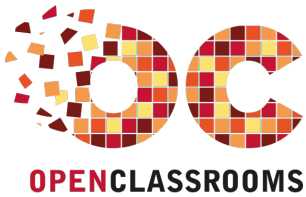
CodeIgniter, le framework au service des Zéros

Par Nestea
et Triviak



www.openclassrooms.com

Licence Creative Commons 6 2.0
Dernière mise à jour le 8/01/2013

Sommaire

Sommaire 2
Lire aussi 2
CodeIgniter, le framework au service des Zéros 4
Partie 1 : Introduction 4
La notion de framework 5
Vous avez dit framework ? 5
Introduction 5
Choisir son framework 5
Frameworks et fonctionnalités 6
Le framework CodeIgniter 6
Présentation 6
Libraries et Helpers 6
En résumé 6
Présentation de l'architecture MVC 6
Découverte de l'architecture MVC 7
Le contrôleur 7
Les vues 7
Les modèles 8
Résumé 8
Le tableau 8
Le diagramme 9
Installation et configuration 9
Téléchargement et installation 10
Téléchargement 10
Installation 10
Mise en place de l'arborescence 11
Les dossiers application et system 11
Le dossier assets 12
Le dossier application 12
Configurons CodeIgniter 12
Le fichier config.php 12
Le fichier database.php 16
Le fichier autoload.php 16
Partie 2 : Mise en place des cinq composants 18
Le contrôleur 18
Votre première page 18
Premier contrôleur 18
Explication des URL 18
En résumé 19
Fonctionnalités du contrôleur 19
Créez d'autres pages 19
Les variables $_GET 19
Organisez vos contrôleurs dans des dossiers 20
Un contrôleur orienté objet 20
Méthodes spéciales 21
La méthode index 21
Le constructeur 22
La méthode _output 23
La méthode _remap 24
La vue 24
Votre première vue 25
Création de la vue 25
Chargement de la vue 25
Transférer vos variables 25
Quelques fonctionnalités sur les vues 27
Gestion des dossiers 27
Charger plusieurs vues 27
Sauvegarder le contenu de vos vues 27
Les helpers 28
Charger et utiliser les helpers 29
Charger un helper 29
Utilisation du helper url 29
Fonction site_url 30
Fonction redirect 30
D'autres fonctions 31
Création du helper « assets » 31
Fonction css_url 32
Fonction js_url 32
Fonction img_url 32
Fonction img 32
Notre helper assets 33
Modifier un helper 33
Créer le fichier 33
Modification de la fonction site_url 34

Ajout de la fonction url ... 35
Notre fichier MY_url_helper.php ... 36
Les bibliothèques ... 37
Utiliser une bibliothèque ... 38
Charger une bibliothèque ... 38
La bibliothèque Session ... 40
Fonctionnement des sessions de CodeIgniter ... 40
Ajouter des éléments ... 41
Supprimer des éléments ... 41
Déconnexion ... 42
Flashdata ... 42
La bibliothèque Form Validation ... 42
Définir les règles des champs ... 43
Récupérer vos valeurs ... 44
Remettre les valeurs dans les champs et afficher les erreurs ... 45
La bibliothèque Database ... 45
Retour sur la fonction site_url ... 46
1re ligne ... 46
2e ligne ... 47
En résumé ... 47
Le modèle ... 47
Anatomie d'un modèle ... 48
Création du modèle ... 48
Utiliser le modèle ... 48
Renommer le modèle ... 49
Le modèle de gestion des news ... 50
Le point sur Active Record ... 51
Méthode ajouter_news ... 51
Méthode modifier_news ... 53
Méthode supprimer_news ... 54
Méthode count ... 55
Méthode liste_news ... 56
Notre modèle de news ... 56
[TP] Un livre d'or ... 59
Description du livre d'or ... 59
Petit rappel ... 59
Descriptif des pages ... 59
Avant de commencer ... 59
Coup de pouce ... 59
Schéma de la table ... 59
Le système de pagination ... 63
Correction ... 64
Analyse et conception ... 64
Le modèle ... 65
Le contrôleur et les vues ... 68
Partie 3 : Utilisations avancées ... 74
L'outil Profiler ... 75
Activer le profiler ... 75
Afficher le profiler ... 75
Réaliser des tests de performance ... 76
Modifier l'ordre des différentes catégories ... 77
Introduction ... 77
Réécrire les méthodes de CodeIgniter : la théorie ... 77
Réécrire les méthodes de CodeIgniter : la pratique ... 78
Ajouter les sessions ... 80
Modification de la méthode run ... 80
Implémentation de la méthode _compile_session ... 81
Modifier le design du profiler ... 84
Modification de la classe Model : le CRUD ... 85
La théorie ... 86
Création du prototype ... 87
Implémentation des méthodes ... 88
La méthode count ... 88
Le méthode create ... 89
La méthode update ... 90
La méthode delete ... 91
La méthode read ... 91
Mise en place des thèmes ... 93
Présentation de l'idée ... 93
Implémentation des méthodes view et views ... 94
Différence entre les deux méthodes ... 94
Implémentation de la méthode views ... 94
Implémentation de la méthode view ... 95
Les variables du layout ... 95
Ajouter le CSS et le JavaScript ... 98
Les méthodes ajouter_css et ajouter_js ... 98
Affichage des liens dans le layout ... 99
Modifier son thème ... 99
Mise en place de l'attribut ... 100
Modification de la méthode view ... 100
Mise en place de la méthode set_theme ... 100
Remerciements ... 101

CodeIgniter, le framework au service des Zéros

Par  Triviak et  Nestea

Mise à jour : 08/01/2013
Difficulté : Difficile Durée d'étude : 15 jours

BY-NC-SA

Vous en avez assez de repartir de zéro à chaque nouveau projet ? Vous voulez développer rapidement et sans contrainte ? Vous souhaitez utiliser un *framework* réputé pour ses performances sans avoir à le configurer pendant une semaine, ni devoir respecter sans arrêt des conventions ?

Eh bien, ami Zéro, je vous accueille à bras ouverts dans ce tutoriel sur CodeIgniter. Nous y découvrirons comment employer ce framework pour réaliser vos sites. Au programme : architecture MVC, programmation orientée objet, bibliothèques, *helpers* et bien d'autres choses !

Pour suivre ce tutoriel, vous devez :

- avoir lu le tutoriel de M@teo21 ;
- avoir lu le tutoriel de vyk12 sur la programmation orientée objet ;
- éventuellement, connaître l'architecture MVC (ce sera un plus).

C'est parti !

Partie 1 : Introduction

Cette partie est essentiellement composée de théorie. Je vous recommande de lire attentivement ces trois chapitres, car ils sont très importants pour bien comprendre ce que nous allons étudier tout au long de ce tutoriel.

Nous verrons, dans un premier temps, la définition d'un framework et les avantages de CodeIgniter. Ensuite, nous nous intéresserons à l'architecture MVC qui nous permettra de structurer notre application web. Enfin, nous téléchargerons CodeIgniter pour l'initialiser ensuite afin de le préparer aux parties suivantes.

La notion de framework

Dans ce chapitre, nous allons définir ce qu'est un framework. Ensuite, nous nous attarderons sur les possibilités de CodeIgniter et ce qui fait de lui un framework très réputé.

Vous allez pouvoir découvrir plus précisément CodeIgniter, mais aussi déterminer si celui-ci est fait pour vous. Chaque framework possède des avantages et des inconvénients. À vous de faire votre choix parmi la multitude de frameworks PHP déjà présents sur le net.

Vous avez dit framework ?

Introduction

C'est bien joli, tout ça, mais tu ne nous as toujours pas dit ce qu'était un framework !

En effet, mais je vais le faire tout de suite.

Citation : Wikipédia

> En programmation informatique, un framework est un kit de composants logiciels structurels, qui définissent les fondations et les grandes lignes de l'organisation de tout ou partie d'un logiciel (architecture).

Autrement dit, un framework PHP est **un ensemble de codes** qui fournit **une organisation** ainsi qu'un grand nombre de **fonctionnalités**, dont le nombre et la qualité diffèrent selon les frameworks. Ainsi, la maîtrise d'un framework vous permet de ne vous occuper que de votre site et de laisser le reste à d'autres développeurs, c'est-à-dire sa base, son socle, mais aussi tout ce qui s'articule autour : les classes, les fonctions, etc.

Choisir son framework

Choisir un framework n'est pas une chose aisée : il en existe un nombre important, dont au moins une bonne dizaine sont de très bonne qualité.

Mais alors, comment choisir ?

Là, c'est une affaire de goût. Cependant plusieurs paramètres sont à prendre en compte. Il n'y a pas que ceux-ci, mais c'est déjà un bon début.

- **Votre style de programmation** : êtes-vous plutôt procédural ou orienté objet ? En PHP, comme dans tout langage de programmation, chacun a son style.
- **Votre organisation** : avis à tous les Zéros bordéliques, c'est peut-être la fin de vos ennuis. La conception de l'organisation diffère d'un codeur à l'autre. Certains préfèrent avoir une arborescence profonde et stricte, d'autres préfèrent la possibilité de placer leurs fichiers n'importe où, comme bon leur semble.
- **Les fonctionnalités** mises en avant par le framework : chacun propose un nombre plus ou moins important de fonctionnalités. Ne tombez pas dans le piège de l'usine à gaz ! Choisissez votre framework raisonnablement. Cela ne sert à rien de choisir un framework trop sophistiqué si vous n'utilisez pas toute sa puissance (ou du moins une partie conséquente).
- **La communauté** et **la réputation** : en effet, quoi qu'on en dise, si le framework est connu, c'est souvent un gage de qualité. Une communauté active, c'est la garantie que le framework continuera à fournir des mises à jour.

Cette liste n'est pas exhaustive. Ce sont simplement des pistes de réflexions.

Frameworks et fonctionnalités

Voici quelques exemples de fonctionnalités que j'ai pu rencontrer dans des frameworks PHP qui peuvent être décisifs :

- ORM pour simplifier la gestion des bases de données ;
- internationalisation ;
- aide pour l'AJAX ;
- gestion des droits (administrateur, modérateur, membre, etc.) ;
- gestion de formats variés : XHTML, XUL, RSS, ATOM, RDF, ZIP, XML, PDF, etc. ;
- et bien d'autres encore.

Certaines peuvent paraître indispensables, alors que d'autres vous sembleront inutiles. Comme je l'ai dit dans l'introduction, c'est à vous de faire votre choix !

Le framework CodeIgniter

Présentation

Maintenant que vous savez ce qu'est un framework, nous allons pouvoir enfin parler de CodeIgniter.

Avant toute chose, il faut savoir que CodeIgniter n'est pas une usine à gaz. Et pour cause, il a été conçu dans le but de ne fournir que le **strict minimum**. Tout le reste est entièrement **optionnel** (même les bibliothèques gérant les bases de données et les sessions le sont).

Cela lui permet donc d'être doublement rapide. D'une part, sur le temps d'apprentissage du framework : vous verrez que vous vous sentirez très vite à l'aise. D'autre part, sur le temps de génération de votre page. Autant vous le dire tout de suite, CodeIgniter se retrouve souvent en très bonne position dans les *benchmarks* (je ne pourrai pas vous en dire plus sur ce point, au risque de ne plus être objectif).

Libraries et Helpers

Cela dit, j'espère que vous ne croyez pas que ce tutoriel se finira au bout de trois malheureux chapitres ! Le socle initial permet aux codeurs d'inclure des bibliothèques et des fonctions fournies par CodeIgniter. Et vous allez le voir, elles sont nombreuses. Vous trouverez la liste ici.

La colonne ***Class Reference*** regroupe des classes que vous pourrez utiliser lorsque vous en aurez besoin. Vous y trouverez notamment les plus importantes :

- **config** pour récupérer toute votre configuration. Vous allez pouvoir personnaliser votre framework ;
- **database** pour les bases de données. Nous la découvrirons en douceur, car elle est assez conséquente ;
- **form_validation** pour simplifier la vérification des formulaires. Une fois que l'on y touche, on ne peut plus s'en passer ;
- **session** pour les sessions. CodeIgniter n'utilise pas les sessions natives mais son propre système. Je vous rassure, ce n'est pas plus compliqué pour autant.

Bien entendu, je ne vous montre qu'une fraction des bibliothèques. Sachez aussi qu'il est possible de créer ses propres classes et même de modifier le comportement de celles fournies par CodeIgniter. Vous verrez, c'est extrêmement utile.

Si vous retournez sur ce lien, vous verrez une colonne ***Helper Reference***. Les helpers sont tout simplement des ensembles de fonctions regroupées par thème. Ici encore, nous allons pouvoir créer nos propres helpers et même redéfinir les fonctions de CodeIgniter lorsqu'elles ne nous plairont pas. Nous avons donc des fonctions pour générer nos URL, générer nos formulaires, insérer des smileys ou encore utiliser un catpcha. Mais pas de panique, il ne s'agit que de simples fonctions.

En résumé

Si l'on devait résumer le framework en une phrase, on dirait que CodeIgniter est une base réduite en fonctionnalités mais hautement performante, pouvant faire appel à des classes et à des fonctions très complètes lorsque le besoin s'en fait sentir. Maintenant que vous en savez plus sur CodeIgniter, nous allons pouvoir nous intéresser à l'organisation de nos fichiers avec l'architecture MVC.

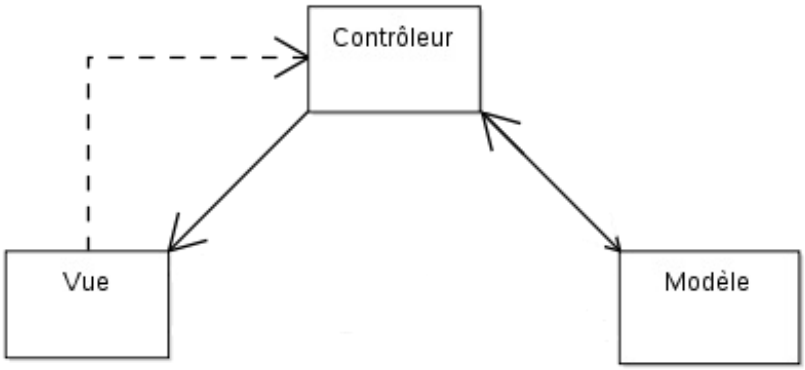
Présentation de l'architecture MVC

Si vous développez en PHP depuis un certain temps, vous avez sûrement dû entendre parler de l'architecture MVC. Non ? Eh bien, si ce n'est pas le cas, vous apprendrez très vite à vous en servir. Vous verrez, ce n'est pas spécialement compliqué, il suffit d'être méthodique.

Si vous connaissez cette architecture, lisez tout de même ce chapitre. Il se peut que vous y trouviez des informations intéressantes.

Découverte de l'architecture MVC

MVC est l'acronyme de **Modèle-Vue-Contrôleur** ! Autrement dit, nous allons séparer notre application en au moins trois parties.



Mais pourquoi ?

Eh bien parce que cela va nous permettre de mieux nous **organiser** !
Il est vrai qu'au début, vous n'en verrez pas tout de suite l'intérêt. Mais lorsque votre site web commencera à prendre de l'ampleur, vous devrez par exemple changer de design, changer le SGBD de votre base de données ou encore ajouter un groupe VIP. Toutes ces modifications annoncent de très lourds changements si vous ne savez exactement où se situe le code à modifier.

Nous allons justement aborder une organisation robuste composée de trois couches d'abstraction.

Le contrôleur

Le contrôleur est ce qui va être appelé en premier. Il n'y a qu'un seul contrôleur par URL. Donc, deux URL différentes appelleront deux contrôleurs différents.

Le contrôleur se charge d'**appeler tous les composants** : bibliothèques, helpers, vues, modèles, mais aussi de **faire les nombreuses vérifications nécessaires**.

Voici quelques exemples de fonctionnalités que peut fournir un contrôleur.

- Vérifier si le formulaire a bien été rempli.
- Vérifier si le visiteur a bien le niveau requis pour voir la page (administrateur par exemple).
- Changer le design selon les préférences du visiteur (via un cookie, par exemple).
- Contrôler les *tokens* (ou jetons) de sécurité. Si vous ne connaissez pas ces petites bêtes, ce n'est pas grave (tout du moins pour ce tutoriel).

Le contrôleur sera donc la partie la plus importante pour le développeur PHP. Tout ce qui se passe dans le contrôleur sera invisible aux yeux des visiteurs. Le contrôleur n'a pas pour rôle d'envoyer au navigateur un message d'erreur, il a pour rôle de trouver cette erreur. L'affichage de cette erreur se fera via une autre couche d'abstraction.

Avec CodeIgniter, vous verrez que chaque contrôleur est une classe qui est appelée selon l'URL. Et comme toute classe qui se respecte, elle possède un certain nombre de méthodes qui représenteront les différentes **actions** que le contrôleur peut effectuer.

Par exemple, si on a cette URL : http://nomdedomaine.tld/news/voir_news.html, cela signifie que CodeIgniter doit appeler dans la classe News la méthode voir_news. C'est donc l'URL qui appelle le contrôleur.

Les vues

Les vues sont directement responsables de **tout ce que l'on va envoyer aux navigateurs**. Les vues seront donc composées principalement de HTML. Nous y trouverons aussi du PHP qui nous servira pour afficher des variables et faire quelques boucles.

Il n'y a pas de limite au nombre de vues. Alors qu'avec les contrôleurs, nous avions une URL par contrôleur, ici, nous pouvons très bien avoir cinq vues par contrôleur. Le rôle du contrôleur est aussi d'appeler la bonne vue. De plus, rien n'empêche le contrôleur de faire appel à plusieurs vues.

Cette relation est représentée sur le schéma par un trait plein allant du contrôleur à la vue.

Sur ce même schéma, vous pouvez voir un trait en pointillé : il s'agit d'un flux indirect. Il est possible, au lieu d'afficher la vue au moment où le contrôleur l'appelle, de sauvegarder son contenu dans une variable, en vue de le stocker dans un fichier par exemple. Ce n'est utile que dans certains cas très particuliers.

Par exemple, pour la page d'accueil de l'administration d'un site, nous pourrons trouver ce genre de vues :

- une vue contenant le formulaire de connexion à la zone administrateur ;
- une vue contenant un message d'erreur ;
- une vue contenant la page d'administration.

Le rôle de la vue s'arrête là. C'est au contrôleur de savoir quelle est la vue qu'il doit charger.

Les modèles

Cette dernière couche d'abstraction permet de faire **le lien entre le contrôleur et la base de données**. Il est toujours conseillé de séparer convenablement les requêtes que vous envoyez à la base de données et les instructions qui vont utiliser ces données.

Le modèle permettra donc d'envoyer des requêtes à la base de données et d'en retourner le résultat sous forme brute (sans HTML). Nous y trouverons donc des fonctionnalités typiques des bases de données (ajouter, modifier, supprimer...).
Dans la plupart des cas, si les bases de données ont bien été pensées, il y aura un modèle pour chaque table.

Prenons l'exemple d'une table sauvegardant des news. Le modèle associé à cette table devra être composé de ces fonctionnalités :

- ajouter ;
- modifier ;
- supprimer ;
- nombre_news ;
- liste_news ;
- news.

Eh bien, pour réaliser cela avec CodeIgniter, vous allez devoir créer une classe que nous appellerons News_model, et nous implémenterons toutes les méthodes dont nous aurons besoin pour manipuler les news dans la base de données. Il nous suffira alors d'exécuter les méthodes de ce modèle depuis le contrôleur.

Pour en finir avec les modèles, il faut que vous sachiez qu'il est facultatif. Vous avez la possibilité de n'en inclure aucun. Ce n'est pas obligatoire de charger un modèle dans le cas où vous ne souhaitez pas exécuter de requêtes avec votre base de données. Mais nous aurons l'occasion d'en parler plus en détail par la suite.

Résumé

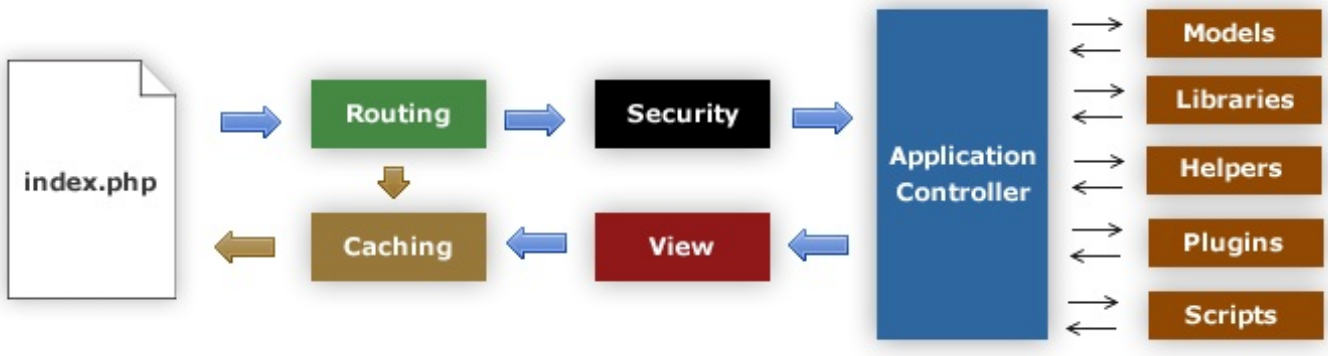
Le tableau

Dorénavant, quand vous développerez avec CodeIgniter, vous devez avoir le tableau suivant dans la tête.

Nom de la couche	Rôle de la couche
Contrôleurs	C'est le cœur de votre application. Ce sont eux qui seront appelés en premier. Ils seront l'intermédiaire entre les modèles, les vues et toute autre forme de ressources.
Vues	Principalement composées de code HTML, elles permettent de renvoyer le code aux visiteurs.
Modèles	Facultatifs, ils permettent d'envoyer des requêtes à la base de données.
Bibliothèques	La plupart sont facultatives. Ce sont des classes qui permettent de nous aider. La bibliothèque de vérification des formulaires est un très bon exemple.
Helpers	Ce ne sont que des fonctions classées par thème.

Le diagramme

Voici un schéma très bien fait extrait de la documentation. Il illustre le fonctionnement de CodeIgniter.



Ordre	Nom	Description
1	index.php	Ce sera toujours le fichier `index.php`, situé à la racine du répertoire, qui sera appelé en premier.
2	Routing	C'est le routeur. C'est lui qui récupérera l'URL et la décomposera en actions. Il a pour rôle de trouver le contrôleur à appeler. Nous verrons dans la suite de ce tutoriel que nous pouvons modifier son comportement.
3	Caching	Une fois que le routeur a fini son travail, le module de cache va regarder s'il existe des fichiers mis en cache pour cette action. Dans le cas d'une réponse positive, ces fichiers vont être renvoyés au navigateur.
4	Security	Cette partie va sécuriser toutes les données entrantes : cookies, variables get, post, etc. C'est la dernière étape avant le contrôleur.
5	Application Controller	Ce sera ici que nous commencerons à développer.
6	Models Libraries Helpers	Comme vous le voyez, le contrôleur fait appel à différents éléments qui vont leur renvoyer des données.
7	View	Les vues sont les fichiers qui contiennent le code HTML.

Si vous êtes attentifs, vous avez dû voir que je n'ai pas du tout parlé des *plugins*. En effet, les plugins ont été supprimés depuis CodeIgniter 2.0, mais l'image n'en reste pas moins très intéressante.

Et voilà, maintenant que vous connaissez le framework et ses grandes lignes, nous allons enfin pouvoir le télécharger et le configurer. J'espère que ces deux chapitres ne vous ont pas trop ramollis, parce qu'à partir de maintenant, il n'y aura plus de chapitre théorique. Ce sera à vous de travailler !

Installation et configuration

Voici enfin venu le moment où vous allez pouvoir télécharger CodeIgniter et ouvrir votre éditeur de texte.

Au menu : téléchargement, installation, découverte de l'arborescence et configuration. À table !

Nous ne toucherons pas à toutes les clés de configuration. Vous comprendrez davantage d'éléments lorsque vous serez habitués au framework.

Téléchargement et installation

Téléchargement

Rendez-vous sur le site officiel de CodeIgniter pour pouvoir télécharger le framework, et plus exactement dans la rubrique downloads. Vous devriez normalement voir un gros bouton *Download CodeIgniter*. À l'heure où j'écris ces lignes, la version actuelle est la 2.0.2. Bien entendu, prenez la dernière version.

Pendant que j'y suis, j'en profite aussi pour vous donner le lien de la documentation. S'il y a bien un lien qu'il faut mettre dans vos favoris, c'est celui-ci.

Une fois qu'elle sera téléchargée, il faudra décompresser l'archive et placer le dossier portant le même nom que la version dans votre répertoire web. Vous pouvez le placer où vous voulez. Pour les besoins du tutoriel, j'ai renommé le dossier en codeIgniter et je l'ai placé dans mon répertoire www.

Installation

Vous allez voir, l'installation va être très rapide.

Quelques modifications

Dans votre dossier (que j'ai appelé codeIgniter), vous avez normalement deux fichiers : index.php et licence.txt. Vous pouvez faire ce que bon vous semble de la licence.
Ensuite, vous avez la documentation dans le dossier user_guid. Vous pouvez aussi en faire ce que bon vous semble.

index.php

Maintenant, vous allez ouvrir le fichier index.php. La première chose à configurer est de définir si le framework est actuellement en période de développement, de test ou de production. La conséquence immédiate est l'affichage ou non des erreurs. Normalement, la constante est déjà sur development.

Code : PHP - ./index.php

```php
<?php

define('ENVIRONMENT', 'development');    // (development | testing | production)
```

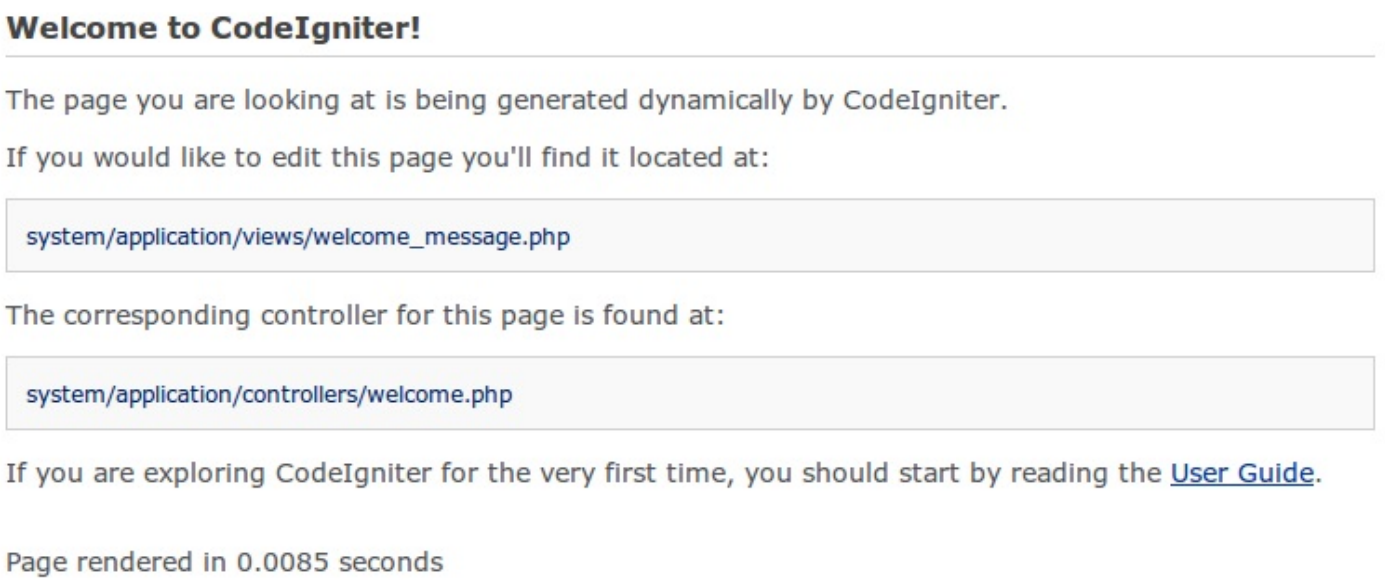
Une fois cela fait, vous pouvez modifier les deux variables suivantes : system_path et application_folder. Le premier est le chemin vers le dossier system et le second est celui vers le dossier application.
La configuration par défaut est fonctionnelle. On pourra par la suite les changer sans problème.

Code : PHP - ./index.php

```php
<?php

/*
|---------------------------------------------------------------
| SYSTEM FOLDER NAME
|---------------------------------------------------------------
*/
 $system_path = "system";

/*
|---------------------------------------------------------------
| APPLICATION FOLDER NAME
|---------------------------------------------------------------
*/
 $application_folder = "application";
```

La documentation de CodeIgniter recommande de renommer ces dossiers pour éviter à un visiteur mal intentionné de connaître leur emplacement. Je pense que vous pouvez vous en passer dans un premier temps.

Vous avez fini d'installer CodeIgniter. Vous avez normalement accès à la page d'accueil du framework (chez moi, c'est http://localhost/codeIgniter/) et celui-ci vous souhaite la bienvenue.



Si vous ne voyez pas ce message, reprenez bien les étapes dans l'ordre et assurez-vous de bien avoir les prérequis de votre version téléchargée (version minimale de PHP, droits de lecture...).

Mise en place de l'arborescence

Maintenant que le framework est installé, attardons-nous sur l'arborescence de CodeIgniter.

Les dossiers application et system

À côté de votre index.php, vous avez deux dossiers : application et system. Le dossier system contient **tous les codes natifs de CodeIgniter** alors que le dossier application va contenir **tous vos codes**. Vous ne devrez donc jamais modifier les fichiers présents dans le dossier system, et je vais vous en expliquer la raison tout de suite.

Imaginons que vous n'aimez pas le nom d'une méthode. Vous allez donc lancer une petite recherche et modifier le nom de cette méthode. Sauf que vous allez avoir de gros problèmes maintenant. En voici trois qui méritent d'être cités.

- Si une nouvelle version de CodeIgniter sort, vous allez devoir refaire vos manipulations. Et ne dites pas que vous ne ferez pas la mise à jour, car vous risquez des problèmes de sécurité.
- En modifiant les sources, vous êtes susceptibles de laisser passer des failles, car il y a de fortes chances que vous ne

connaissiez que l'aspect « utilisateur » du framework.
- Un des avantages de l'utilisation d'un framework est la facilité de travailler à plusieurs, car chacun sait comment le système fonctionne. Mais si vous modifiez les sources de CodeIgniter, son comportement ne sera plus le même.

Nous verrons par la suite comment effectuer ces changements sans toucher à ce dossier.

Le dossier assets

Avant de décrire précisément le contenu du dossier application, vous allez créer un dossier assets, toujours dans le même dossier que le fichier index.php. Et à l'intérieur de celui-ci, créez aussi les dossiers css, javascript et images.

Vous disposez maintenant d'une arborescence très claire.

- **assets** contiendra toutes les ressources qui vont être téléchargées par les visiteurs. Par exemple, les feuilles de style externes (CSS), les fichiers JavaScript et les images. Cependant, vous pouvez aussi ajouter des dossiers tels que musiques, videos, xml (pour les flux RSS, les animations flash). C'est comme si vous alliez faire un copier-coller de ce dossier dans l'ordinateur de vos visiteurs. Rien ne doit être confidentiel.
- **system** contient les sources de CodeIgniter, vous ne devez pas y toucher.
- **application** contiendra tous vos scripts, qu'il faudra placer dans les bons dossiers. C'est ce que nous allons voir tout de suite.

Le dossier application

Voici l'arborescence du dossier application avec les explications des dossiers dont nous nous servirons dans la prochaine partie.

- **config** : ce sont des fichiers permettant de configurer CodeIgniter ou une bibliothèque. Certains sont inclus automatiquement, d'autres seulement lorsque vous les demandez.
- **controllers** : ce dossier contiendra tous nos contrôleurs.
- **errors** : ce sont les pages d'erreurs. Libre à vous de les personnaliser selon votre design.
- **helpers** : c'est le répertoire pour vos helpers. Les helpers fournis avec CodeIgniter sont situés dans le dossier system.
- **hooks** : c'est un dossier qui contient des fichiers assez complexes. Ils permettent d'exécuter des scripts à différents moments du processus d'exécution de CodeIgniter.
- **language** : ce répertoire contiendra tous vos fichiers de langue dans le cas où vous souhaitez un site internationalisé.
- **libraries** : nous placerons nos bibliothèques dans ce dossier. Comme les helpers, les bibliothèques fournies par CodeIgniter sont situées dans un autre dossier.
- **models** : le répertoire des modèles.
- **views** : le répertoire des vues.

En respectant les emplacements des différents fichiers, vous gagnerez en clarté.

Configurons CodeIgniter

Certes, l'installation est terminée, mais il reste encore certains points à définir.

Le fichier `config.php`

Ce fichier de configuration est le cœur de la configuration du framework. Il se trouve dans le dossier `./application/config/`. Je ne vais pas décrire tous les points mais sachez que nous allons aborder les plus importants.

`base_url`

C'est l'URL que vous devez taper pour accéder au fichier `index.php`. Dans mon cas, ce sera :

Code : PHP - ./application/config/config.php

```
<?php
```

```
/*
|-------------------------------------------------------------------
-------
| Base Site URL
|-------------------------------------------------------------------
-------
*/
$config['base_url'] = "http://localhost/codeIgniter/";
```

index_page

Par défaut, la valeur est index.php. Nous allons voir que CodeIgniter génère par défaut des URL de cette forme-là : http://localhost/codeIgniter/index.php/classe_controleur/methode_controleur/.

Problème, ce « index.php » n'est pas des plus jolis. Pour le masquer, il y a une solution : l'*URL rewriting*. Si vous avez activé ce module, alors vous pouvez créer un fichier `.htaccess` à côté du `index.php` contenant ce code :

Code : Apache - ./.htaccess

```
# Empêche la visualisation de l'arborescence, n'a rien à voir avec
le masquage du « index.php ».
Options -Indexes

# Active le module de réécriture d'URL.
RewriteEngine on

#
# Fixe les règles de réécriture d'URL. Ici, nous utilisons une
liste blanche.
#

# Toutes les URL qui ne correspondent pas à ces masques sont
réécrites.
RewriteCond $1 !^(index\.php|assets/|robots\.txt)

# Toutes les autres URL vont être redirigées vers le fichier
index.php.
RewriteRule ^(.*)$ index.php/$1 [L]
```

Une fois cela fait, vous pouvez vider la variable `index_page` puisque les URL seront redirigées vers ce fameux fichier `index.php`.

Code : PHP - ./application/config/config.php

```
<?php

/*
|-------------------------------------------------------------------
-------
| Index File
|-------------------------------------------------------------------
-------
*/
// Dans le cas où mod_rewrite est activé
$config['index_page'] = "";

// Dans le cas contraire
$config['index_page'] = "index.php";
```

Maintenant, vos URL ne contiendront plus le « index.php ».

url_suffix

C'est **le suffixe** que vous voulez donner à vos URL. Vous pouvez donner n'importe quoi. CodeIgniter l'ignorera...
Dans mon cas, je vais choisir l'extension html. Vous pourrez tout aussi bien ne rien mettre. C'est facultatif.

Code : PHP - ./application/config/config.php

```php
<?php

/*
|--------------------------------------------------------------------------
| URL suffix
|--------------------------------------------------------------------------
*/

$config['url_suffix'] = ".html";
```

language

Cette option permet de configurer **la langue par défaut**. C'est très utile si vous voulez afficher les erreurs en français et non en anglais.

Cette langue sera aussi utilisée pour les messages d'erreurs lorsque vos formulaires ne seront pas bien remplis. Si vous voulez faire un site en français, prenez le temps de faire cette manipulation.

Si c'est le cas, alors il va falloir modifier certaines choses. Dans un premier temps, passez cette variable à « *french* ».

Code : PHP - ./application/config/config.php

```php
<?php

/*
|--------------------------------------------------------------------------
| Default Language
|--------------------------------------------------------------------------
*/
$config['language'] = "french";
```

Désormais, lorsqu'il y aura une erreur, CodeIgniter voudra l'afficher en français. Sauf que CodeIgniter ne propose qu'un jeu de langue anglais... Jetez un petit coup d'œil aux fichiers situés dans le répertoire `./system/language/`. Il n'y a que le dossier `english`.

Vous avez donc deux solutions : traduire l'ensemble des fichiers de langue ou récupérer un fichier de langue sur le web.

Si vous voulez récupérer un fichier de langue, vous pourrez en trouver sur le wiki officiel ou bien dans les sources de PyroCMS. C'est un CMS qui a été développé avec CodeIgniter. Il vous suffit de télécharger ce CMS, de le décompresser et d'aller dans le répertoire `./codeigniter/language/`. Dans ce dossier, vous avez accès à une panoplie de langages, dont le français (*french*). Il vous suffit donc de déplacer le dossier `french` de PyroCMS jusque dans votre répertoire `./system/language/`.

Maintenant, CodeIgniter est en mesure d'afficher les erreurs en français.

session

Vous pouvez aussi configurer l'utilisation des **sessions**.

Voici ce que vous pouvez paramétrer :

Code : PHP - ./application/config/config.php

```php
<?php

// Le nom du cookie...
$config['sess_cookie_name'] = 'ci_session';

// La date de péremption du cookie, en secondes...
$config['sess_expiration'] = 7200;
```

En plus de cela, CodeIgniter propose de stocker les sessions dans votre base de données au lieu de les stocker dans les cookies.

Si vous stockez souvent beaucoup de données dans les sessions, choisissez évidemment la base de données. Dans la plupart des cas, je vous conseille de prendre cette option car elle offre en plus une sécurité identique aux sessions natives de PHP.

Si vous êtes intéressés par cette fonctionnalité, alors vous devez dans un premier temps créer une table *ci_sessions* dans votre base de données.
Ce code vous permettra de générer la table. Il s'agit d'un simple code SQL.

Code : SQL - Création de la table ci_session

```sql
CREATE TABLE IF NOT EXISTS `ci_sessions` (
 session_id      varchar(40)     DEFAULT '0' NOT NULL,
 ip_address      varchar(16)     DEFAULT '0' NOT NULL,
 user_agent      varchar(120)    NOT NULL,
 last_activity   int(10)         unsigned DEFAULT 0 NOT NULL,
 user_data       text            NOT NULL,

 PRIMARY KEY (session_id),

 KEY `last_activity_idx` (`last_activity`)
);
```

Une fois ceci fait, vous devez lui dire d'utiliser la base de données.

Code : PHP - ./application/config/config.php

```php
<?php

// La valeur TRUE permet d'utiliser la base de données
$config['sess_use_database'] = TRUE;

// Le nom de la table
$config['sess_table_name'] = 'ci_sessions';
```

compress_output

Cette clé permet d'activer la compression Gzip. Elle est désactivée par défaut car tous les hébergeurs ne supportent pas cette fonctionnalité. Pour rappel, cette compression permet de réduire la taille du contenu envoyé au navigateur. En théorie, les pages devraient se charger plus rapidement.

Lorsque que vous êtes en développement, désactivez cette fonctionnalité car si vous ne respectez pas les standards du MVC, votre page ne s'affichera pas. Il est courant d'utiliser la fonction `var_dump` un peu n'importe où. Cependant, si vous affichez des données à l'extérieur des vues, l'affichage de votre page plantera.

Maintenant que cela est fait, nous allons configurer notre base de données.

Le fichier database.php

Dans ce fichier, vous noterez vos identifiants et le mot de passe de connexion à votre base de données.

Code : PHP - ./application/config/database.php

```php
<?php

/*
| -------------------------------------------------------------------
| DATABASE CONNECTIVITY SETTINGS
| -------------------------------------------------------------------
*/

$db['default']['hostname'] = "nom_d_hote";
$db['default']['username'] = "nom_d_utilisateur";
$db['default']['password'] = "mot_de_passe";
$db['default']['database'] = "base_de_donnees";
```

Maintenant, votre base de données est opérationnelle.

Vous avez la possibilité d'inscrire plusieurs bases de données. En effet, la base de données d'exemple utilise la ligne « `default` ». Vous pouvez ajouter d'autres lignes dans la tableau `$db` pour gérer tous vos identifiants.

Le fichier autoload.php

Ce fichier vous permet d'inclure dès le lancement de CodeIgniter des bibliothèques, des *helpers*, ainsi que des modèles et fichiers de langue.

Bibliothèque

Je vous propose de charger au démarrage du framework deux bibliothèques : les sessions et les bases de données. Je trouve que ce sont des fonctionnalités « élémentaires » qui nécessitent d'être constamment utilisées ou presque.

Pour faire cela, vous devez ajouter dans le tableau les bibliothèques que vous voulez charger.

Code : PHP - ./application/config/autoload.php

```php
<?php

/*
| -------------------------------------------------------------------
| Auto-load Libraries
| -------------------------------------------------------------------
*/
$autoload['libraries'] = array('database', 'session');
```

Lorsque vous connaîtrez plus de bibliothèques, ou que vous aurez besoin d'inclure les vôtres, vous pourrez les ajouter à la suite.

Helpers

Nous allons aussi charger un helper : `url`. Il nous permettra d'écrire des URL en fonction des paramètres que vous aurez définis dans le fichier `./application/config/config.php`. En utilisant ces fonctions, toutes vos URL seront instantanément modifiées si la valeur de `$config['base_url']` est modifiée.

Code : PHP - ./application/config/autoload.php

```php
<?php

/*
| -------------------------------------------------------------------
| Auto-load Helper Files
| -------------------------------------------------------------------
*/

$autoload['helper'] = array('url');
```

Nous en avons fini avec la configuration de CodeIgniter.
Cette partie est terminée. Nous allons maintenant attaquer la deuxième partie.

Comme je vous l'ai dit dans la partie précédente, c'est le contrôleur qui est appelé en premier. Nous allons donc aborder celui-ci dès maintenant.

Partie 2 : Mise en place des cinq composants

La partie traitant de l'installation et de la configuration est maintenant derrière vous ! Vous allez pouvoir réellement découvrir le framework.

L'utilisation d'un framework n'étant pas évidente, je vous conseille de prendre l'habitude de faire régulièrement des tests chez vous. Vous mémoriserez bien plus facilement.

Le contrôleur

Dans cette partie, nous allons créer notre première page avec CodeIgniter.

Pour rappel, le contrôleur est ce qui sera chargé en premier dans votre application. C'est lui qui fera les vérifications. Il aura accès aux sessions et à toutes les données transitant entre le navigateur et le serveur : get, post, cookie, etc.

Votre première page

Premier contrôleur

Je vois que vous bouillez d'impatience de créer votre premier contrôleur, alors le voici :

Code : PHP - ./application/controllers/forum.php

```php
<?php

class Forum extends CI_Controller
{
 public function accueil()
 {
  echo 'Hello World!';
 }
}
```

Dans CodeIgniter, un contrôleur est un fichier contenant uniquement une classe. Tout se passera donc dans cette classe.

Écrivez cette classe dans un fichier forum.php et placez celui-ci dans le dossier ./application/controllers/. Si vous vous rendez à l'adresse http://localhost/codeIgniter/index.php/forum/accueil/, vous verrez la page « Hello World! »
Veillez à donner le bon chemin dans l'URL dans le cas où vous avez placé le framework dans un autre dossier que le mien.

Normalement, vous ne devez jamais afficher du texte depuis le contrôleur. Cela marche souvent très bien, sauf que vous ne respectez pas l'architecture MVC. Nous verrons dans le chapitre suivant comment créer des vues.

Explication des URL

L'URL par défaut

Les URL sont formées à partir des contrôleurs en suivant ce modèle :
http://www.nomdedomaine.tld/index.php/classe_controleur/methode_controleur/.

C'est le cas dans l'exemple ci-dessus. Notre classe s'appelle Forum et elle comporte une méthode accueil. Cela se traduit donc par une URL contenant ces segments : http://www.nomdedomaine.tld/index.php/forum/accueil.

L'URL avec une extension

Si vous avez bien suivi le chapitre précédent, vous aviez la possibilité de donner une extension aux URL. Il s'agit de la clé de configuration url_suffix du fichier ./application/config/config.php. Pour ma part, je l'avais initialisée avec l'extension html.

Dans ce cas, mon URL deviendra : http://www.nomdedomaine.tld/index.php/forum/accueil.html.

L'URL sans le index.php

Dans le chapitre précédent, je vous ai dit que vous pouviez activer l'*URL rewriting* pour masquer l'appel au fichier index.php. Si vous avez fait cela, votre URL sera alors : http://www.nomdedomaine.tld/forum/accueil.html.

Rassurez-vous, il existe des fonctions toutes prêtes pour construire nos URL !

En résumé

Un contrôleur est donc un fichier qui contient une unique classe et ce sera son nom qui s'affichera dans l'URL.

Pour que CodeIgniter puisse savoir où est situé le contrôleur à appeler, vous devez nommer votre fichier du même nom que votre classe mais sans la majuscule. De plus, le nom de la classe doit toujours commencer par une majuscule.

Fonctionnalités du contrôleur

Créez d'autres pages

Vous avez la possibilité de créer d'autres méthodes qu'`accueil`.
Exemple avec ce contrôleur :

Code : PHP - ./application/controllers/forum.php

```php
<?php

class Forum extends CI_Controller
{
 public function accueil()
 {
  echo 'Hello World!';
 }

 public function bonjour()
 {
  echo 'Salut à tous !';
 }

 public function manger()
 {
  echo 'Bon appétit !';
 }
}
```

Désormais, votre contrôleur possède plusieurs pages qui seront accessibles de la même manière :

- `forum/bonjour.html`
- `forum/manger.html`

Chaque méthode permet donc de faire apparaître une page. Si avant cela, vous n'aviez jamais utilisé de framework, ce doit déjà être une petite révolution.

Les variables `$_GET`

Maintenant que vous avez créé vos pages et que vous connaissez les URL pour y accéder, vous voulez peut-être utiliser les variables `$_GET`. Eh bien, pas de souci. On n'aurait pas pu faire plus simple !

Code : PHP - ./application/controllers/forum.php

```php
<?php
```

```php
class Forum extends CI_Controller
{
 public function accueil()
 {
  echo 'Hello World!';
 }

 // Cette page accepte une variable $_GET facultative
 public function bonjour($pseudo = '')
 {
  echo 'Salut à toi : ' . $pseudo;
 }

 // Cette page accepte deux variables $_GET facultatives
 public function manger($plat = '', $boisson = '')
 {
  echo 'Voici votre menu : <br />';
  echo $plat . '<br />';
  echo $boisson . '<br />';
  echo 'Bon appétit !';
 }
}
```

Ces URL seront tout à fait correctes :
forum/accueil/
forum/bonjour/Arthur.html
forum/manger/Hamburger-frites/coca.html

Alors, ce n'est pas joli, la POO ?

Ainsi, votre site web se découpera en contrôleurs, qui auront chacun des méthodes. L'ensemble formera votre site. Voici un exemple :

User	Accueil	Forum
accueil	accueil	accueil
inscription	faq	categorie
desinscription	reglement	voir_post
connexion	plan	voir_message
deconnexion	contact	
profil		

Si vous aviez déjà pris l'habitude de « découper » votre site en modules (news, forum, livre d'or...), alors la migration vers CodeIgniter sera rapide. En revanche, si vous aviez opté pour quelque chose d'un peu plus exotique, dites-vous qu'au moins votre site sera dorénavant plus organisé. 

Organisez vos contrôleurs dans des dossiers

Dans le cas où vous créez un gros projet qui nécessite de nombreux contrôleurs, CodeIgniter vous autorise **à les placer dans des dossiers**. Cependant, vous vous doutez bien que ce n'est pas aussi simple que cela.
Si, par exemple, vous voulez placer 3 contrôleurs dans un dossier « forum », alors votre URL devra afficher le paramètre « forum » en premier, avant le nom du contrôleur :
forum/contrôleur/méthode/

Un contrôleur orienté objet

N'oubliez pas que le contrôleur n'est pas uniquement un moyen de classer ses pages ! Il s'agit avant tout d'une classe.

Autrement dit, vous pouvez lui donner **des attributs** et même **des méthodes privées**. Si, dans votre contrôleur, vous devez exécuter régulièrement les mêmes instructions, créez une méthode privée pour éviter la redondance de votre code. Si votre méthode est privée, elle ne sera pas accessible par l'URL (tout comme les méthodes commençant par un *underscore* _).

Méthodes spéciales

CodeIgniter a prévu pour vous certaines particularités dans les contrôleurs. Nous allons les voir tout de suite.

La méthode index

Cette méthode a la particularité d'être appelée dans le cas où votre URL ne spécifie pas de nom de méthode.

Un exemple est toujours plus parlant :

Code : PHP - ./application/controllers/forum.php

```php
<?php

class Forum extends CI_Controller
{
 public function index()
 {
  echo 'Index';
 }

 public function accueil()
 {
  echo 'Hello World!';
 }
}
```

Lorsque vous irez sur ces trois URL, vous appellerez donc la méthode index.

- forum/
- forum.html avec l'extension.
- forum/index.html

Vous verrez donc s'afficher le message « Index ».

Astuce

N'oubliez pas que vous êtes dans une classe. Rien ne vous empêche de faire ceci :

Code : PHP - application/controllers/forum.php

```php
<?php


class Forum extends CI_Controller
{
 public function index()
 {
  $this->accueil();
 }

 public function accueil()
 {
  echo 'Hello World!';
 }
}
```

Mon exemple est trop simple, car vous remplacez une ligne par une autre. Mais dans le cas d'une méthode accueil assez importante, cela peut être intéressant.

J'ai appelé cette technique « redirection interne », car elle est invisible depuis un navigateur.

Le constructeur

Encore une méthode intéressante, le constructeur. Dans les contrôleurs, vous utiliserez cette méthode pour effectuer des tâches d'initialisation. Autrement dit, vous allez placer tout le code qui devra être exécuté à chaque fois que vous allez sur l'une des pages de ce contrôleur. C'est particulièrement utile si vous devez initialiser des attributs ou charger des bibliothèques et des helpers (que nous verrons plus loin).

Une petite remarque : si vous implémentez le constructeur, vous devez **absolument** exécuter le constructeur de la classe mère avant de commencer votre code.

Exemple :

Code : PHP - ./application/controllers/forum.php

```php
<?php

class Forum extends CI_Controller
{
 private $titre_defaut;

 public function __construct()
 {
  // Obligatoire
  parent::__construct();

  // Maintenant, ce code sera exécuté chaque fois que ce contrôleur
sera appelé.
  $this->titre_defaut = 'Mon super site';
  echo 'Bonjour !';
 }

 public function index()
 {
  $this->accueil();
 }

 public function accueil()
 {
  var_dump($this->titre_defaut);
 }
}
```

Si vous retournez sur une page de ce contrôleur, vous verrez donc le message « Bonjour ! » et la variable sera initialisée.

Mon exemple est complètement nul, je le sais. Mais j'en ai un autre. L'authentification est prévue pour un autre chapitre, mais vous savez que les contrôleurs doivent faire attention aux différents niveaux d'authentification. Il pourrait exister par exemple un module entièrement réservé aux administrateurs. Pour cela, il suffira de placer une condition dans le constructeur pour que toutes les pages de celui-ci soient protégées.

Code : PHP - application/controllers/admin.php

```php
<?php

class Admin extends CI_Controller
{
 public function __construct()
 {
  // Obligatoire
  parent::__construct();
```

```
        if( ! isAdmin())
        exit("Vous n'avez pas le droit de voir cette page.");
         }

         public function activer_maintenance()
         {
          /* ---- */
         }

         public function ajouter_rang()
         {
          /* ---- */
         }
        }
```

Cela permettra de faire rapidement le tri entre les administrateurs et les autres. À vous d'utiliser cette méthode comme bon vous semble.

La méthode _output

La méthode _output vous permet de manipuler une dernière fois les données que vous allez envoyer au navigateur.

Cette fonction n'est pas accessible depuis le navigateur, car elle est précédée d'un « _ » (underscore).

Output en anglais signifie « sortie ».

Code : PHP - ./application/controllers/forum.php

```
        <?php

        class Forum extends CI_Controller
        {
         public function __construct()
         {
          // Obligatoire
          parent::__construct();
         }

         public function index()
         {
          $this->accueil();
         }

         public function accueil()
         {
          echo 'Bonjour';
         }

         // L'affichage de la variable $output est le comportement par
        défaut.
         public function _output($output)
         {
          var_dump($output);
         }
        }
```

Si vous effectuez quelques tests, vous verrez que cette méthode n'a aucun effet parce que **le contrôleur ne doit pas afficher de texte**. Nous verrons bientôt l'utilisation des vues. Vous pourrez l'utiliser ensuite.

La méthode _remap

_remap est une méthode très puissante. Elle vous permet de modifier la méthode que vous allez utiliser. Autrement dit, faire une redirection interne un peu plus dans le style de CodeIgniter. Nous allons voir un exemple d'utilisation de cette fonctionnalité mais nous l'étudierons plus en détail dans le chapitre portant sur le routeur.

Voici un exemple d'utilisation. Nous allons rediriger toutes les URL du contrôleur vers la page de maintenance.

Code : PHP - ./application/controllers/home.php

```php
<?php

class Home extends CI_Controller
{
 public function accueil()
 {
  echo 'Bonjour';
 }

 public function maintenance()
 {
  echo "Désolé, c'est la maintenance.";
 }

 public function _remap($method)
 {
  $this->maintenance();
 }
}
```

Quelle que soit l'URL, ce sera toujours la méthode maintenance qui sera appelée. Il s'agira donc d'une redirection interne en continu...

À partir du moment où vous allez implémenter cette méthode, vous devrez le faire jusqu'au bout. Si vous n'y mettez rien, alors aucune méthode ne sera appelée. Vous devrez donc faire la vérification de l'existence de la page. Si vous souhaitez afficher la page d'erreur 404, vous pouvez utiliser la méthode *show_404()* pour l'afficher.

C'est tout !

Vous connaissez désormais tout ce qu'il faut savoir sur les contrôleurs. Le contrôleur n'est pas un élément très complexe, mais il ne vous permet pas de faire de vraies pages web à la sauce CodeIgniter.

Voyons donc tout de suite comment créer et afficher les vues !

La vue

Vous savez maintenant comment créer des contrôleurs avec CodeIgniter. Maintenant, nous allons aborder les vues.

Les vues sont composées de code HTML et d'un peu de PHP. Ce sont les données qui sont envoyées au navigateur.

Votre première vue

Création de la vue

Pour créer une vue, vous devez tout d'abord créer un fichier dans le dossier `./application/views/`. Placez-y tout ce que vous voulez.

Dans mon cas, je vais écrire ce contenu dans un fichier `vue.php` :

Code : PHP - ./application/views/vue.php

```
<h1>Bonjour</h1>

<p>
    Ceci est mon paragraphe !
</p>
```

Rien de particulier. Un titre et un paragraphe.

Chargement de la vue

Lorsque vous avez fini de créer votre vue, vous devez l'afficher *via* le contrôleur. Voici comment il faut faire.

Code : PHP - ./application/controllers/news.php

```
<?php
class News extends CI_Controller
{
 public function __construct()
 {
  parent::__construct();
 }

 public function index()
 {
  $this->accueil();
 }

 public function accueil()
 {
$this->load->view('vue');
 }
}
```

Cette ligne vous permet d'afficher le contenu de la vue. Le premier paramètre attend donc le nom du fichier sans l'extension.

Transférer vos variables

Pour transférer des variables du contrôleur à la vue, nous allons dans un premier temps créer un tableau qui contiendra toutes les données à transmettre à la vue.

Code : PHP - ./application/controllers/news.php

```
<?php

class News extends CI_Controller
```

```
{
 public function index()
 {
  $this->accueil();
 }

 public function accueil()
 {
  $data = array();
  $data['pseudo'] = 'Arthur';
  $data['email'] = 'email@ndd.fr';
  $data['en_ligne'] = true;

$this->load->view('vue');
}
}
```

Maintenant que nous avons notre tableau de variables, nous allons pouvoir le donner à la vue. Pour cela, il faut utiliser le 2e argument de la méthode *view*.

Code : PHP - ./application/controllers/news.php

```
<?php

class News extends CI_Controller
{
 public function index()
 {
  $this->accueil();
 }

 public function accueil()
 {
  $data = array();
  $data['pseudo'] = 'Arthur';
  $data['email'] = 'email@ndd.fr';
  $data['en_ligne'] = true;

  // Maintenant, les variables sont disponibles dans la vue
  $this->load->view('vue', $data);
 }
}
```

Je vais donc modifier ma vue pour y faire apparaître les variables.

Les variables sont retirées de leur tableau. Depuis la vue, la variable `$data` n'existera pas. En revanche, les variables `$pseudo`, `$email` et `$en_ligne` seront bien présentes.

Code : PHP - ./application/views/vue.php

```
<h1>Bonjour</h1>

<p>
    Ceci est mon paragraphe !
</p>

<p>
    Votre pseudo est <?php echo $pseudo; ?>.
</p>

<p>
```

```
    Votre email est <?php echo $email; ?>.
</p>

<p>
<?php if($en_ligne): ?>
    Vous êtes en ligne.
<?php else: ?>
    Vous n'êtes pas en ligne.
<?php endif; ?>
</p>
```

Maintenant, vous savez comment transférer des variables du contrôleur à la vue.

Quelques fonctionnalités sur les vues

Gestion des dossiers

Vous avez la possibilité de créer des dossiers dans le répertoire `views`. Une bonne habitude consiste à classer ses vues en fonction du contrôleur. Autrement dit, vous allez placer toutes les vues du contrôleur `News` dans le dossier `./application/views/news/`.

Pour appeler ces vues, vous devrez donner le chemin relatif avant le nom de la vue à afficher.

Code : PHP - Un contrôleur

```
<?php

$this->load->view('news/vue');
```

Je vous fais confiance pour organiser convenablement vos vues. N'hésitez pas à créer des dossiers.

Charger plusieurs vues

Certaines personnes n'y pensent pas toujours, j'ajoute donc une petite partie uniquement pour elles. Vous pouvez charger autant de vues que vous voulez.

Code : PHP

```
<?php

public function accueil()
{
 $this->load->view('theme/en_tete');
 $this->load->view('theme/menu_gauche');
 $this->load->view('theme/menu_droit');

 $this->load->view('news/accueil');

 $this->load->view('theme/footer');
}
```

Si vous devez constamment charger des vues, vous avez la possibilité de les placer dans le constructeur du contrôleur. Toutes les instructions seront exécutées et cela compte aussi pour les vues. Nous verrons dans une autre partie comment inclure automatiquement l'en-tête et le pied-de-page. Donc ne vous ennuyez pas trop avec vos thèmes et vos layouts.

Sauvegarder le contenu de vos vues

Le problème avec la méthode `view`, c'est qu'elle affiche tout le temps le contenu de la vue. Alors si vous voulez sauvegarder dans un fichier le contenu de ce qui va s'afficher (faire un système de cache), vous serez bloqués.

Pour résoudre cela, vous pouvez utiliser le 3e argument de la méthode `view`. C'est un booléen. Si sa valeur est `false` (défaut), alors la vue sera affichée. Par contre, si la valeur est `true`, alors cette méthode retournera le contenu de la vue mais ne l'affichera pas.

Code : PHP - ./application/controllers/

```php
<?php

public function accueil()
{
 $data = array;
 $data['pseudo'] = 'Arthur';
 $data['email'] = 'email@ndd.fr';
 $data['en_ligne'] = true;

$vue = $this->load->view('vue', $data, true);
}
```

Vous savez maintenant créer des contrôleurs et des vues. N'hésitez pas à construire un petit site web pour effectuer vos tests. Avec les contrôleurs et les vues, il y a déjà pas mal à faire.

Nous n'aborderons pas les modèles dès le prochain chapitre, car nous n'avons pas encore appris à lancer des requêtes à la base de données. Nous allons donc d'abord nous préparer en découvrant les helpers et les bibliothèques, puis nous finirons par les modèles.

Les helpers

Comme je vous l'ai dit dans la partie I, les helpers constituent un ensemble de fonctions réunies par thème. Vous n'allez donc pas charger une fonction à la fois mais un ensemble de fonctions.

Nous allons donc voir comment charger un helper. Ensuite, nous apprendrons à créer les nôtres et même à modifier les helpers natifs.

Charger et utiliser les helpers

Pour charger un helper, vous allez voir, c'est très simple. Et pour les utiliser, c'est encore plus simple, car il ne s'agit que de simples fonctions. J'espère que vous êtes au point avec les fonctions...

Charger un helper

Pour charger un helper depuis un contrôleur, nous allons utiliser cette instruction :

Code : PHP - Dans un contrôleur

```php
<?php

$this->load->helper('nom');
```

Si vous voulez charger plusieurs helpers en même temps, il est possible de passer en paramètre un tableau au lieu d'une chaîne de caractères.

La liste des helpers est disponible ici, dans la colonne de droite.

Utilisation du helper url

Nous allons apprendre à nous servir du helper url.

Commençons par le charger :

Code : PHP - Dans un contrôleur

```php
<?php

$this->load->helper('url');
```

À partir de maintenant, vous pourrez utiliser les fonctions de ce helper dans votre contrôleur, mais aussi dans votre vue.

Voici la documentation de ce helper.

Si vous aviez entré dans votre autoloader (./application/config/autoload.php) le nom de ce helper, alors vous n'avez pas besoin de le charger avant de l'utiliser.

Voici un contrôleur et une vue classique pour pouvoir tester ces fonctions :

Code : PHP - ./application/controllers/test.php

```php
<?php

class Test extends CI_Controller
{
 public function __construct()
 {
```

```
        parent::__construct();

        // « Décommenter » cette ligne pour charger le helper url
        //$this->load->helper('url');
    }

    public function accueil()
    {
        // On inclut la vue ./application/views/test/accueil.php
        $this->load->view('test/accueil');
    }
```

Code : PHP - ./application/views/test/accueil.php

```
<h1>
    Test
</h1>
```

Fonction site_url

Nous allons commencer par la fonction la plus importante : site_url.
Cette fonction permet de retourner une URL. C'est donc avec cette fonction que nous allons créer toutes les URL à partir de maintenant.

Pourquoi ?

Car cela va rendre vos URL beaucoup moins dépendantes de votre environnement, et donc beaucoup plus flexibles !

En utilisant cette fonction, vous ne devrez plus donner votre nom de domaine, ni le fichier index.php (imaginez que vous changez d'hébergeur et qu'il n'y ait plus l'URL rewriting), ni l'extension. Vous êtes libres de modifier toutes vos URL en même temps grâce à cette fonction.
Désormais, je l'utiliserai constamment pour poursuivre ce tutoriel.

Il y a deux façons de construire son URL : avec une chaîne de caractères et avec un tableau.

Code : PHP - ./application/views/test/accueil.php

```
<h1>
 Test
</h1>

<p>
 <a href="<?php echo site_url(); ?>">accueil</a>
 <br />

 <a href="<?php echo site_url('test'); ?>">accueil</a> du test
 <br />

 <a href="<?php echo site_url('test/secret'); ?>">page secrète</a>
 <br />

 <a href="<?php echo site_url(array('test', 'secret')); ?>">page
secrète</a>
</p>
```

Fonction redirect

La fonction redirect fonctionne de la même manière que site_url. Elle permet d'effectuer une redirection de type « location ». Elle accepte en paramètre une chaîne de caractères ou un tableau.

Code : PHP - ./application/controllers/test.php

```php
<?php
class Test extends CI_Controller
{
 public function __construct()
 {
  parent::__construct();

  // Décommenter cette ligne pour charger le helper url
  //$this->load->helper('url');
 }

 public function index()
 {
redirect(array('error', 'probleme'));
 }

 public function accueil()
 {
  $this->load->view('test/accueil');
 }
```

D'autres fonctions

current_url	Cette fonction renvoie l'URL de la page actuelle.
base_url	Cette fonction renvoie votre URL, mais sans les segments (variables GET, nom du contrôleur, nom de la méthode...). Dans la plupart des cas, ce sera votre nom de domaine. Par exemple, sur le SdZ, cette fonction retournera http://www.siteduzero.com/

Je ne vous ai présenté que les fonctions les plus importantes de ce helper. Je vous laisse découvrir les autres dans la documentation.

L'utilisation d'un helper est relativement simple. Vous le chargez et vous utilisez ses fonctions. Sauf que durant votre développement, vous pourrez avoir besoin d'ajouter des fonctions aux helpers, voire d'en créer d'autres. C'est ce que nous allons faire dès maintenant.

Création du helper « assets »

Nous allons maintenant réaliser un helper **`assets`**. Celui-ci permettra de récupérer plus rapidement les URL des fichiers contenus dans le dossier `assets`. Il comportera quatre fonctions.

Prototype	Description
`string` **`css_url`** `( string $nom )`	Permettra de retourner l'URL d'un fichier CSS.
`string` **`js_url`** `( string $nom )`	Permettra de retourner l'URL d'un fichier JavaScript.
`string` **`img_url`** `( string $nom )`	Permettra de retourner l'URL d'une image.
`void` **`img`** `( string $nom , string $alt = '')`	Permettra de créer rapidement le code HTML des images.

Pour créer un nouveau helper, il faut créer un fichier dans le dossier `./application/helpers/` . Nous l'appellerons `assets_helper` (oui, le suffixe est obligatoire). Ensuite, il faut l'inclure *via* l'`autoloader` ou manuellement : `<?php $this->load->helper('assets');`

Afin de ne pas compliquer inutilement le tutoriel, ce helper utilisera des fonctions du helper URL. Vous devrez donc

veiller à l'inclure soit *via* l'*autoloader*, soit depuis votre contrôleur avant de l'utiliser.

Fonction `css_url`

Cette fonction attend un paramètre qui devra être une chaîne de caractères. Il nous suffit donc simplement d'ajouter le nom de domaine avant, puis le chemin, ensuite le paramètre et enfin l'extension.
Le nom de domaine sera récupéré via la fonction `base_url`.

Eh bien allons-y !

Code : PHP - ./application/helpers/assets_helper.php

```php
<?php

function css_url($nom)
{
 return base_url() . 'assets/css/' . $nom . '.css';
}
```

Cette petite fonction nous permettra de déplacer très facilement le dossier `assets` sans réécrire toutes les URL pour inclure les fichiers CSS.

Fonction `js_url`

Rien de nouveau dans cette fonction. C'est quasiment la même que `css_url`.

Code : PHP - ./application/helpers/assets_helper.php

```php
<?php

function js_url($nom)
{
 return base_url() . 'assets/javascript/' . $nom . '.js';
}
```

Fonction `img_url`

Il y a un petit piège dans cette fonction, mais je suis sûr que vous ne tomberez pas dedans !

Code : PHP - ./application/helpers/assets_helper.php

```php
<?php

function img_url($nom)
{
 return base_url() . 'assets/images/' . $nom;
}
```

En effet, cette fonction ne peut pas connaître à l'avance l'extension de l'image, contrairement aux fichiers CSS et JavaScript.

Fonction `img`

Cette fonction attend deux paramètres. Le premier est le nom de l'image. Nous allons donner cette variable directement à la fonction `img_url`. Le deuxième paramètre sera pour l'attribut `alt` de l'image. Il est donc facultatif.

Code : PHP - ./application/helpers/assets_helper.php

```php
<?php

function img($nom, $alt = '')
{
 return '<img src="' . img_url($nom) . '" alt="' . $alt . '" />';
}
```

Notre helper assets

Voici donc notre helper `assets`. J'ai encadré toutes les fonctions avec une condition pour éviter de redéfinir des fonctions. Nous allons voir tout de suite après qu'il y a aussi une autre raison à cela.

Code : PHP - ./application/helpers/assets_helper.php

```php
<?php  if ( ! defined('BASEPATH')) exit('No direct script access
allowed');

if ( ! function_exists('css_url'))
{
 function css_url($nom)
 {
  return base_url() . 'assets/css/' . $nom . '.css';
 }
}

if ( ! function_exists('js_url'))
{
 function js_url($nom)
 {
  return base_url() . 'assets/javascript/' . $nom . '.js';
 }
}

if ( ! function_exists('img_url'))
{
 function img_url($nom)
 {
  return base_url() . 'assets/images/' . $nom;
 }
}

if ( ! function_exists('img'))
{
 function img($nom, $alt = '')
 {
  return '<img src="' . img_url($nom) . '" alt="' . $alt . '" />';
 }
}
```

La première ligne du fichier permet d'exécuter le script seulement si la constante `BASEPATH` est définie. En fait, elle est créée au tout début du fichier `index.php`. En faisant cela, vous serez assurés que le script ne sera pas exécuté depuis l'URL, mais bien en suivant l'ordre normal des choses.

Désormais, vous savez comment créer entièrement un helper. Cela permettra de diminuer grandement la redondance de votre application. Maintenant, nous allons voir comment redéfinir et même ajouter quelques fonctions à un helper natif.

Modifier un helper

Créer le fichier

Une fonction ne vous convient pas dans un helper ? Vous avez envie de rajouter une fonction dans un helper ? Voyons tout de suite comment nous allons faire.

Retournez dans votre fichier de configuration (config.php) et cherchez cette ligne :

Code : PHP - ./application/config/config.php

```php
<?php

$config['subclass_prefix'] = 'MY_';
```

Elle vous indique le préfixe du nom que doit avoir votre fichier pour modifier le comportement d'un script natif de CodeIgniter. Si vous appelez votre fichier MY_url_helper.php, vous allez pouvoir réécrire les fonctions du helper url. En fait, notre fichier va être lancé avant le helper natif. Nous y définissons des fonctions telles que site_url, ou current_url. Puis une fois arrivé au bout du fichier, CodeIgniter appellera le helper natif.

Mais nous allons définir deux fonctions avec le même nom, alors ?

Non ! Car si vous vous souvenez de la partie précédente, chaque fonction est entourée d'une condition :

Code : PHP - ./system/helpers/url_helper.php

```php
<?php

if ( ! function_exists('site_url'))
{
 function site_url($uri = '')
 {
  /* ****** */
 }
}
```

Donc nous définissons les fonctions que nous voulons changer dans notre fichier MY_url_helper.php, puis CodeIgniter se chargera d'inclure toutes les autres fonctions que nous n'avons pas réécrites.

Créons donc notre fichier MY_url_helper.php dans le dossier ./application/helpers/.

Modification de la fonction site_url

Nous allons modifier la fonction site_url. Pourquoi ? Parce que j'ai envie de lui offrir la possibilité de lui passer autant de paramètres que l'on souhaite.

Code : PHP - Dans un contrôleur ou une vue

```php
<?php

$url_secret = site_url('forum', 'secret', '2452', 'codeigniter');
/*
$url_secret va donc valoir quelque chose comme cela :
http://nom_de_domaine.tld/forum/secret/2452/codeigniter.html
*/
```

L'implémentation est assez simple, car nous allons seulement modifier l'en-tête de la fonction. Nous ne toucherons pas au reste. Voici la fonction d'origine :

Code : PHP - ./system/helpers/url_helper.php

```php
<?php

if ( ! function_exists('site_url'))
{
 function site_url($uri = '')
 {
  $CI =& get_instance();
  return $CI->config->site_url($uri);
 }
}
```

Vous ne comprendrez sûrement pas le code de cette fonction, car elle fait appel à des notions que nous allons voir immédiatement après ce chapitre. En fait, cette fonction est délocalisée. Les instructions ne sont pas dans la fonction mais dans une autre partie de CodeIgniter.
Dans notre cas, nous ne modifierons pas cette partie.

Voici ce que nous allons faire : si le premier paramètre est un tableau, nous ne modifions rien. Mais si le premier paramètre est une chaîne de caractères, alors nous allons créer un tableau avec tous les paramètres envoyés à la fonction. Cela s'implémente très facilement avec la fonction func_get_args qui permet de retourner sous forme de tableau tous les paramètres envoyés à notre fonction.

Code : PHP - ./application/helpers/MY_url_helper.php

```php
<?php

function site_url($uri = '')
{
 if( ! is_array($uri))
 {
  // Tous les paramètres sont insérés dans un tableau
  $uri = func_get_args();
 }

 // On ne modifie rien ici
 $CI =& get_instance();
 return $CI->config->site_url($uri);
}
```

Désormais, la fonction site_url aura un comportement différent.

Mais pourquoi n'a-t-on pas créé un autre helper ?

Parce que ça n'aurait pas de sens. Vous auriez dû appeler deux helpers pour gérer vos URL. Modifier les helpers natifs de cette façon est une très bonne habitude. Je vous rappelle qu'il est fortement déconseillé de modifier tout code présent dans le dossier ./system/.

Ajout de la fonction url

C'est un peu bête d'avoir fait tout ça pour modifier trois lignes... Je vous ai donc concocté une autre fonction : url. Elle permet de créer rapidement des liens. Elle prendra en premier paramètre le texte à insérer dans le lien. Et en second paramètre, nous créons le lien de la même façon que la fonction site_url.

Code : PHP - Dans un contrôleur ou une vue

```php
<?php

// Affichera (selon les préférences) : <a
href="http://nom_de_domaine.tld/user/connexion.html">Page de
```

```
connexion</a>
url('Page de connexion', 'user', 'connexion');
```

Voici le code de cette fonction.

Code : PHP - ./application/helpers/MY_url_helper.php

```
<?php

function url($text, $uri = '')
{
 if( ! is_array($uri))
 {
  $uri = func_get_args();

  // Suppression de la variable $text
  array_shift($uri);
 }

 echo '<a href="' . site_url($uri) . '">' . htmlentities($text) .
'</a>';
 return '';
}
```

Vous noterez que nous avons utilisé la fonction array_shift en plus. Elle permet d'envoyer à la fonction site_url tous les paramètres que nous avons reçus sauf un : le texte.

Notre fichier MY_url_helper.php

Voici donc notre ficher MY_url_helper.php :

Code : PHP - ./application/helpers/MY_url_helper.php

```
<?php  if ( ! defined('BASEPATH')) exit('No direct script access
allowed');

if ( ! function_exists('site_url'))
{
 function site_url($uri = '')
 {
  if( ! is_array($uri))
  {
   // Tous les paramètres sont insérés dans un tableau
   $uri = func_get_args();
  }

  // On ne modifie rien ici
  $CI =& get_instance();
  return $CI->config->site_url($uri);
 }
}

// ------------------------------------------------------------------
-------

if ( ! function_exists('url'))
{
 function url($text, $uri = '')
 {
  if( ! is_array($uri))
  {
   // Suppression de la variable $text
   $uri = func_get_args();
```

```
        array_shift($uri);
      }

      echo '<a href="' . site_url($uri) . '">' . htmlentities($text) .
'</a>';
      return '';
    }
}


/* End of file MY_url_helper.php */
/* Location: ./application/helpers/MY_url_helper.php */
```

Pourquoi la surcharge du helper url possède-t-elle toujours les conditions d'existence des fonctions ?

Question tout à fait légitime. J'y répondrai par une autre question : et si quelqu'un voulait réécrire les fonctions que nous avons réécrites ?

C'est tordu, non ? En fait, si vous voulez réécrire encore une fois les fonctions, vous n'allez pas passer par un troisième fichier. Vous allez réécrire les fonctions de **votre** fichier, celui qui est dans le dossier *application*.

Donc ça ne sert à rien ?

Non plus ! Je vais reformuler une autre question : et si quelqu'un voulait réécrire les fonctions que nous avons réécrites mais seulement une seule fois ? Bon, je vous l'accorde, je ne l'ai encore jamais fait, mais c'est tout de même plus propre. Ça évite les risques de redéfinir une fonction qui est déjà définie dans un autre helper.
Cette partie vous a permis de bien comprendre la notion de helpers dans CodeIgniter. N'hésitez pas à créer les vôtres si votre code devient trop redondant.

Cependant, l'utilisation de helpers n'est pas l'unique moyen d'éviter la redondance. Dans le prochain chapitre, nous verrons un autre outil : les bibliothèques.

Les bibliothèques

Nous allons aborder ici un gros morceau de Codelgniter : les bibliothèques.. Nous venons de voir les helpers, de petites fonctions qui la plupart du temps sont dans le style « passe-partout ».
Ce n'est pas du tout le cas des bibliothèques. Celles-ci sont là pour nous faire gagner des semaines !

Vous allez pouvoir forger votre propre opinion, car nous allons voir trois bibliothèques : *session*, *form_validation* et *database*.

Utiliser une bibliothèque

Charger une bibliothèque

C'est la même procédure que pour les helpers : avant d'utiliser, il faut charger.

Voici comment faire pour charger une bibliothèque :

Code : PHP - Dans un contrôleur

```php
<?php

$this->load->library('nom_de_la_bibliothèque');
```

Voici un exemple pour que vous compreniez. C'est théorique. Ne le faites pas chez vous.

Voici ma bibliothèque :

Code : PHP - ./application/libraries/alphabet.php

```php
<?php

class Alphabet
{
 private $lettres = 'abcdefghijklmnopqrstuvwxyz';

 public function __construct()
 {

 }

 public function recuperer_alphabet()
 {
  return $this->lettres;
 }

 public function supprimer_alphabet()
 {
  $this->lettres = '';
 }

 public function changer_alphabet($lettres)
 {
  if(is_string($lettres) AND !empty($lettres))
  {
   $this->lettres = $lettres;
   return true;
  }
  else
  {
   return false;
  }
 }
}
```

L'utilité de cette classe est quasi nulle, mais nous ferons avec. Voici le code dans le contrôleur qui nous permettra d'avoir accès à cette bibliothèque.

Code : PHP - contrôleur

```php
<?php

public function accueil()
{
 $data = array();

 // Chargement de notre bibliothèque Alphabet
$this->load->library('alphabet');

 $this->load->view('accueil', $data);
}
```

Maintenant, je vais vous faire une **révélation** : voici comment nous accédons à la méthode *recuperer_alphabet* de cette bibliothèque.

Code : PHP - contrôleur

```php
<?php

public function accueil()
{
 $data = array();

 // Chargement de notre bibliothèque Alphabet
 $this->load->library('alphabet');

$data['alphabet'] = $this->alphabet->recuperer_alphabet();

 $this->load->view('accueil', $data);
}
```

Comme vous le voyez, l'expression $this->alphabet nous permet d'avoir accès à toutes les méthodes et à tous les attributs de notre bibliothèque !

N'y a-t-il pas quelque chose qui vous choque dans ce code ?

À la ligne 20 :
`<?php $this->load->view('accueil', $data);`

Enlevons le superflu :
`<?php $this->load->view();`

C'est **exactement** la même chose que notre alphabet ! L'expression $this->load fait appel aux méthodes de **la bibliothèque Load**.

Lorsque nous chargeons une bibliothèque, un helper, une vue et – c'est l'objet du chapitre suivant – un modèle, nous appelons une méthode de la bibliothèque Load. Si vous ne me croyez pas, allez faire un tour dans le fichier *./system/core/loader.php*. Il s'agit de la bibliothèque Load. Vous comprenez désormais ce bout de code que vous utilisiez aveuglément depuis le début. Il ne s'agissait en fait que d'une bibliothèque.

Nous n'avons pas besoin d'initialiser certaines bibliothèques, car elles font partie du cœur du framework. La bibliothèque Load en fait partie. Il serait difficile d'utiliser CodeIgniter sans rien charger... C'est pour cette raison que certaines bibliothèques sont automatiquement chargées.

Voici un code qui exploite toutes les fonctionnalités de notre bibliothèque.

Code : PHP - contrôleur

```php
<?php

public function accueil()
{
 $data = array();

 // Chargement de notre bibliothèque Alphabet via la bibliothèque
Load
 $this->load->library('alphabet');

 // Appel de la méthode Alphabet::recuperer_alphabet
 $data['defaut_alphabet'] = $this->alphabet->recuperer_alphabet();

 // Appel de la méthode Alphabet::supprimer_alphabet
 $this->alphabet->supprimer_alphabet();

 // Appel de la méthode Alphabet::changer_alphabet
 $data['alphabet'] = 'acegikmoqsuwybdfhjlnprtvxz';
 $this->alphabet->changer_alphabet($data['alphabet']);

 // Appel de la méthode Load::view
 $this->load->view('accueil', $data);
}
```

Accéder aux méthodes de nos bibliothèques est donc extrêmement simple une fois que l'on a compris le principe !

Nous allons maintenant survoler trois bibliothèques.

La bibliothèque Session

Session est une des bibliothèques que nous avons insérées dans l'autoloader (*./application/config/autoload.php*). Nous pourrons donc utiliser directement ses méthodes :

```php
<?php $this->session->method();
```

Si vous ne l'avez pas fait, il vous suffit de la charger, comme nous avons chargé la bibliothèque Alphabet.

Fonctionnement des sessions de CodeIgniter

Je vous l'ai dit dans l'introduction : CodeIgniter possède son propre système de session *via* les cookies ou la base de données. Que vous ayez choisi l'un ou l'autre, il n'y aura aucune différence d'utilisation.

Mais pourquoi n'utilise-t-on pas les sessions natives de PHP ?

Cette bibliothèque permet d'avoir des sessions très configurables. Vous pouvez par exemple modifier très facilement la durée de vie des sessions, ce qui est impossible si vous n'avez pas accès à la configuration de PHP.
Nous verrons aussi de nouvelles fonctions pour manipuler les sessions, comme les sessions temporaires.

Dans votre fichier *./application/config/config.php*, vous trouverez un booléen (sess_encrypt_cookie) qui permet de crypter ou non les données. C'est utile si vous stockez vos données dans des cookies (mais vous atteindrez plus vite la limite de 4 KB). La valeur par défaut est false. Il vous faudra remplir la clé de cryptage dans le fichier de configuration *config.php*.

Pour reconnaître les visiteurs, CodeIgniter leur assigne un identifiant unique qui est stocké dans un cookie. Si vous utilisez une base de données, c'est ce cookie qui servira de repère. De plus, il est régénéré toutes les 5 minutes. Si vous voulez le récupérer, il faudra utiliser la méthode *userdata*.

Code : PHP - Utilisation de la méthode userdata

```php
<?php

$session_id = $this->session->userdata('session_id');
```

CodeIgniter sauvegarde aussi 3 autres informations :

Code : PHP - Récupération des trois autres données

```php
<?php

$adresse_ip                = $this->session->userdata('ip_address');
$user_agent_du_navigateur = $this->session->userdata('user_agent');
$derniere_visite           = $this->session-
>userdata('last_activity');
```

Les sessions ne sont renouvelées que toutes les 5 minutes si vous n'avez pas changé la clé de configuration. Cela signifie que votre variable $derniere_visite ne sera pas actualisée à chaque connexion.

Les sessions ne se limitent pas à cela. Vous avez la possibilité d'y stocker vos propres données.

Ajouter des éléments

La méthode *set_userdata* vous permettra d'ajouter des données dans les sessions. Les sessions fonctionnent de la même manière qu'un tableau associatif. On a une clé sous forme de chaîne de caractères à laquelle on associe une donnée.

Code : PHP - Utilisation de la méthode set_userdata

```php
<?php

$this->session->set_userdata('nom_de_votre_valeur', 'valeur');
```

Un exemple d'utilisation :

Code : PHP - Exemple d'utilisation des méthodes userdata et set_userdata

```php
<?php

$this->session->set_userdata('pseudo', 'Arthur');

// ..............

$pseudo = $this->session->userdata('pseudo');
// $pseudo vaut 'Arthur'
```

Si vous essayez d'accéder à une valeur qui n'existe pas, la méthode *userdata* vous retournera false.

Supprimer des éléments

La méthode *unset_userdata* permet de faire cela. Elle ne supprimera pas toutes les sessions, seulement les clés que vous précisez.

Code : PHP - Utilisation de la méthode user_userdata

```php
<?php

$this->session->unset_userdata(array('pseudo' => '', 'email' =>
''));
```

Déconnexion

Si vous souhaitez supprimer toute la session, alors utilisez la méthode *sess_destroy* :

Code : PHP - Utilisation de la méthode sess_destroy

```php
<?php

public function deconnexion()
{
 // Détruit la session
 $this->session->sess_destroy();

 // Redirige vers la page d'accueil
 redirect();
}
```

Flashdata

Voici une petite fonctionnalité supplémentaire dans les sessions : les sessions temporaires. Il s'agit de données qui ne vont être stockées que pour une seule utilisation.

Méthode	Explication
set_flashdata	Permet d'ajouter une donnée. Fonctionne de la même façon que *set_userdata*.
flashdata	Permet de récupérer les données. Lorsque vous utilisez cette méthode, l'élément stocké en session sera détruit. Fonctionne de la même manière que *usersata*.
keep_flashdata	Même chose que *flashdata* sauf que l'élément ne sera pas détruit une fois récupéré. L'utilisation en continu de cette méthode vous fait perdre l'utilité des flashdatas.

Ces méthodes fonctionnent de la même manière que les méthodes classiques. Je ne vous donne pas d'exemple.

Nous avons vu toute la bibliothèque *session* en détail. Ce n'est pas bien compliqué : on ajoute, on récupère et on détruit. Cela suit le même principe que les sessions natives de PHP. Faites des tests chez vous pour bien assimiler !

La bibliothèque Form Validation

Nous allons maintenant nous intéresser à une autre bibliothèque qui vous sera très utile, car il s'agit d'un système très simple (du côté utilisateur, en tout cas ) pour vérifier si un formulaire est valide ou non.

Le principe est le suivant : vous créez votre formulaire avec votre vue ; ensuite, vous définissez des règles pour chaque champ de votre formulaire. Par exemple, vous définissez votre champ « pseudo » comme une chaîne alphanumérique, ou bien votre champ « téléphone » comme un entier positif de 10 chiffres, puis vous demandez à la bibliothèque de vérifier si le formulaire est valide ou non. C'est extrêmement utile, car vous allez simplifier bon nombre de vérifications.

Nous allons tester cette bibliothèque avec un formulaire classique de connexion.

Code : PHP - La vue

```
<form method="post" action="">
 <label for="pseudo">Pseudo : </label>
 <input type="text" name="pseudo" value="" />

 <label for="mdp">Mot de passe :</label>
 <input type="password" name="mdp" value="" />

 <input type="submit" value="Envoyer" />
</form>
```

Cette vue fait apparaître deux champs texte et un bouton.

Et voici le contrôleur de test :

Code : PHP

```
<?php

public function connexion()
{
 // Chargement de la bibliothèque
 $this->load->library('form_validation');

 if($this->form_validation->run())
 {
  // Le formulaire est valide
  $this->load->view('connexion_reussi');
 }
 else
 {
  // Le formulaire est invalide ou vide
  $this->load->view('formulaire');
 }
}
```

Voici comment nous commençons à utiliser la bibliothèque de formulaire. Nous la chargeons, puis nous coupons en deux notre contrôleur. La méthode *run* retournera true **uniquement** si votre formulaire est valide. Étant donné que nous n'avons pas encore défini de règles pour chaque champ, cette méthode retournera constamment false. Voyons donc comment définir nos règles.

Définir les règles des champs

Nous définissons les règles de nos champs avec la méthode *set_rules*.

- Le premier paramètre est le nom du champ, celui qui est contenu dans l'attribut *name* de la balise.
- Le deuxième attribut est le nom « humain » du champ. Par exemple, le nom « humain » du champ « pseudo » est « Nom d'utilisateur ». C'est cette valeur qui sera affichée à l'utilisateur pour lui indiquer le champ qu'il doit corriger.
- Le troisième paramètre est une chaîne de caractères où nous allons définir nos règles. Nous y plaçons des mots clés séparés par des pipes (des barres verticales : |). On peut accéder à la liste de toutes les options disponibles sur cette page.

Vous allez voir, c'est très intuitif.

Code : PHP - contrôleur

```
<?php

public function connexion()
{
 // Chargement de la bibliothèque
 $this->load->library('form_validation');
```

```
$this->form_validation->set_rules('pseudo', '"Nom d\'utilisateur"',
'trim|required|min_length[5]|max_length[52]|alpha_dash|encode_php_tags|xss_clean');
$this->form_validation->set_rules('mdp', '"Mot de passe"',
'trim|required|min_length[5]|max_length[52]|alpha_dash|encode_php_tags|xss_clean');

 if($this->form_validation->run())
 {
  // Le formulaire est valide
  $this->load->view('connexion_reussie');
 }
 else
 {
  // Le formulaire est invalide ou vide
  $this->load->view('formulaire');
 }
}
```

Décomposons ensemble cette chaîne de caractères :
trim|required|min_length[5]|max_length[52]|alpha_dash|encode_php_tags|xss_clean.

- **trim** : n'importe quelle fonction PHP native qui n'attend qu'un seul paramètre de type string peut être indiquée ici. Elle sera appliquée au contenu du champ du formulaire. Ici, on applique la fonction trim (qui permet de retirer tous les espaces au début et à la fin d'une chaîne).
- **required** : permet de définir le champ comme obligatoire. Si l'option *required* n'est pas définie, alors il sera possible de valider le formulaire sans que le champ ne soit rempli.
- **min_length[*entier*]** : permet de définir la longueur minimale du champ. Ici, j'estime que le pseudo ainsi que le mot de passe ne doivent pas avoir une longueur inférieure à 5 caractères. Vous pouvez choisir le nombre que voulez, bien entendu.
- **max_length[*entier*]** : permet de définir la longueur maximale du champ. Cette option fonctionne de la même manière que min_length.
- **alpha_dash** : cette directive permet de n'accepter le champ que si celui-ci contient uniquement des caractères alphanumériques, des underscores (« _ ») et des tirets.
- **encode_php_tags** : convertit les balises PHP en entités HTML.
- **xss_clean** : supprime les risques de failles XSS.

N'hésitez pas à jeter un coup d'œil aux autres directives. Certaines sont très intéressantes, comme *matches* qui permet de ne valider le formulaire que lorsque deux champs sont strictement identiques (pratique pour la confirmation du mot de passe lors d'une inscription, par exemple). Notez que vous pouvez aussi définir vos propres directives.

Maintenant, la validation du formulaire sera fonctionnelle. Cependant, bien que déjà très sympathique, cela ne constitue qu'une partie de cette bibliothèque. Nous allons maintenant découvrir d'autres fonctionnalités.

Récupérer vos valeurs

Vos valeurs pourront être récupérées via la bibliothèque Input. L'utilisation de celle-ci est très simple. Elle comporte quatre méthodes majeures : *post*, *get*, *cookie*, *server*. Ces méthodes nous permettront de récupérer les données entrantes.

Code : PHP

```
<?php

$pseudo = $this->input->post('pseudo');
$mdp = $this->input->post('mdp');
```

Input est une bibliothèque native de CodeIgniter que vous n'avez pas besoin d'inclure. En effet, elle permet aussi de récupérer les variables GET de l'URL. Sans cela, l'utilisation de CodeIgniter serait réduite à une page unique. Comme je l'ai dit dans la présentation de ce tutoriel, CodeIgniter n'inclut que le strict minimum.

Remettre les valeurs dans les champs et afficher les erreurs

Ce serait bête de s'arrêter là. En cas d'erreur, vous pouvez remettre dans les champs les valeurs envoyées et même afficher les erreurs.
Pour cela, nous utiliserons les méthodes *set_value* et *form_error*. La première retourne la valeur et la seconde affiche l'erreur (si erreur il y a).

Code : PHP - La vue

```
<form method="post" action="">
 <label for="pseudo">Pseudo : </label>
 <input type="text" name="pseudo" value="<?php echo
set_value('pseudo'); ?>" />
 <?php echo form_error('pseudo'); ?>

 <label for="mdp">Mot de passe :</label>
 <input type="password" name="mdp" value="" />
 <?php echo form_error('mdp'); ?>

 <input type="submit" value="Envoyer" />
</form>
```

Désormais, les valeurs se rempliront toutes seules et l'erreur s'affichera toute seule aussi.

Cette bibliothèque offre d'autres fonctionnalités comme la création d'options personnalisées (*callback*), le changement des délimiteurs pour l'affichage des erreurs, etc. N'hésitez pas à les découvrir !

La bibliothèque Database

La bibliothèque *Database* vous offre une couche d'abstraction pour réaliser vos requêtes. Autrement dit, la plupart du temps, vous n'allez plus les effectuer avec une chaîne de caractères mais à l'aide de méthodes (celles de la bibliothèque *Database*).

Pour charger la bibliothèque et vous connecter à la base de données, vous devez utiliser la méthode *Database* du loader.

Code : PHP

```
<?php

$this->load->database();
```

Petite particularité ! Pour appeler les méthodes de cette bibliothèque, il vous faut utiliser l'attribut « db » et non « database ».

Si vous êtes au point avec le langage SQL, alors vous n'aurez aucun problème avec cette bibliothèque. Là encore, son utilisation est très intuitive. Voici un exemple de requête que nous pouvons effectuer :

Code : PHP - Utilisation de l'Active Record

```
<?php

$resultat = $this->db->select('id, email')
                     ->from('utilisateurs')
                     ->where('pseudo', 'ChuckNorris')
                     ->limit(1)
                     ->get()
                     ->result();
```

Cette requête se lit comme ceci :
« Sélectionne-moi les colonnes 'id' et 'email' de la table 'utilisateurs' où le champ 'pseudo' vaut 'ChuckNorris' et arrête-toi dès que

tu auras 1 résultat. Merci beaucoup. »

Vous verrez que le fait d'utiliser des méthodes améliorera la lisibilité de votre code et vous permettra de simplifier tout le processus de construction de vos requêtes. De plus, le passage à un autre SGBD en sera simplifié.
Nous aurons l'occasion d'approfondir un peu l'utilisation de la bibliothèque dans le chapitre suivant.

Lorsque nous aurons des requêtes beaucoup plus compliquées, il sera peut-être judicieux de revenir aux requêtes sous forme de chaînes de caractères. Voici un exemple :

Code : PHP - Avec la méthode query

```php
<?php

// Mise en place de notre requête
$sql = "SELECT `id`,
`email`
FROM `utilisateurs`
WHERE `pseudo` = ?
LIMIT 0,1
;";

// Les valeurs seront automatiquement échappées
$data = array('ChuckNorris');

// On lance la requête
$query = $this->db->query($sql, $data);

// On récupère le nombre de résultats
$nb_resultat = $query->num_rows();

// On parcourt l'ensemble des résultats
foreach($query->result() as $ligne)
{
 echo $ligne->id;
}

// On libère la mémoire de la requête (fortement conseillé pour
lancer une seconde requête)
$query->free_result();
```

Je ne peux pas vraiment vous expliquer en détail le fonctionnement de cette bibliothèque. Elle est beaucoup trop consistante. Cependant, elle vous donne accès à énormément de fonctionnalités. Nous verrons par la suite des exemples de requêtes, notamment dans le chapitre sur les modèles (le suivant).

Retour sur la fonction site_url

Vous vous souvenez de notre fonction *site_url* ? Je vous avais dit que vous comprendriez le code plus tard. Eh bien vous êtes en mesure de le comprendre maintenant :

Code : PHP - ./system/helpers/url_helper.php

```php
<?php

function site_url($uri = '')
{
 $CI = get_instance();
 return $CI->config->site_url($uri);
}
```

Décomposons les deux lignes.

1re ligne

La première ligne stocke dans une variable la « valeur » de retour de la fonction *get_instance*. Cette fonction vous permet d'**utiliser vos bibliothèques à l'intérieur de vos fonctions et de vos classes**. En effet, nous savons que pour appeler les méthodes des bibliothèques, vous devez faire quelque chose comme cela :

Code : PHP

```php
<?php

$this->alphabet->supprimer_alphabet();
```

Sauf que votre $this est tout à fait adapté lorsque vous êtes dans les couches MVC ($this est aussi disponible dans les vues). Mais si nous voulons appeler nos bibliothèques depuis une fonction ou même depuis une bibliothèque, nous sommes bloqués. Cette fonction va donc nous aider à contourner le problème. Elle permet de retourner l'instance d'une classe où vous pourrez utiliser les bibliothèques.

2e ligne

La deuxième ligne utilise l'instance que nous donne la fonction pour utiliser une méthode de la bibliothèque Config qui est *site_url*. C'est cette méthode qui fera tout le boulot. C'est pour cela que j'avais dit que cette fonction est délocalisée.

En résumé

Ce qu'il faut retenir de cela, c'est que la valeur de retour de la fonction *get_instance* vous permettra d'avoir accès à vos bibliothèques depuis vos classes et vos fonctions.
Par convention, cette instance est nommée $CI. Si vous ouvrez une bibliothèque native, il y a fort à parier que vous trouviez un attribut $CI qui est initialisé au constructeur par cette méthode.
Nous venons de voir brièvement trois bibliothèques. Mais il en existe beaucoup d'autres.

Maintenant que vous commencez à voir la puissance des bibliothèques, vous comprendrez mieux la définition de CodeIgniter que je vous ai donnée dans la partie introductive :

Citation : Partie I

> Si l'on devait résumer le framework en une phrase, on dirait que CodeIgniter est une base, réduite en fonctionnalités mais hautement performante, pouvant faire appel à des classes et à des fonctions très complètes lorsque le besoin s'en fait sentir.

Le modèle

Vous avez vu les contrôleurs et les vues. Vous avez vu les helpers et les bibliothèques. Il ne vous manque plus que les modèles !

Dans cette partie, nous utiliserons le pattern *Active Record* de la classe database pour effectuer nos requêtes.

Rappel : les modèles servent à faire les requêtes dans la base de données et à retourner le résultat sous forme brute (sans HTML).

Anatomie d'un modèle

Tout au long de ce chapitre, nous allons créer un modèle pour la gestion d'un système de news.

Création du modèle

Créez un fichier news_model.php dans votre dossier ./application/models/.
Voici ce que vous allez y placer. Il s'agit d'un modèle basique.

Code : PHP - ./application/models/news_model.php

```php
<?php  if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class News_model extends CI_Model
{
 public function get_info()
 {
  // On simule l'envoi d'une requête
  return array('auteur' => 'Chuck Norris',
        'date' => '24/07/05',
               'email' => 'email@ndd.fr');
 }
}
```

Le modèle est donc une classe qui hérite de la classe Model et qui contient des méthodes. Dans le cas où vous voulez implémenter le constructeur, il faut obligatoirement appeler le constructeur parent.
La méthode get_info permet de retourner des données pour réaliser vos tests. Par la suite, nous y effectuerons une vraie requête.

La classe doit avoir le même nom que votre fichier, mais avec une majuscule.

Utiliser le modèle

Avant d'utiliser un modèle depuis un contrôleur, il faut le charger via le contrôleur. Pour cela, vous avez la méthode model de la bibliothèque Load.

Code : PHP - Un contrôleur

```php
<?php

class User extends CI_Controller
{
 public function accueil()
 {
  // Chargement du modèle de gestion des news
$this->load->model('news_model');
 }
```

Le fichier news_model.php étant dans le dossier model, CodeIgniter le trouvera tout seul.

Vous pouvez charger votre modèle dans toutes vos pages avec l'autoloader (./application/config/autoload.php).

Vous avez aussi la possibilité de charger le modèle dans le constructeur d'un contrôleur pour y avoir accès dans toutes ses méthodes.

Maintenant que nous avons chargé le modèle, nous allons récupérer les données de la méthode get_info.

Code : PHP - Un contrôleur

```php
<?php
class User extends CI_Controller
{
 public function accueil()
 {
  // Chargement du modèle de gestion des news
  $this->load->model('news_model');

  $data = array();

  // On lance une requête
$data['user_info'] = $this->news_model->get_info();

  // On inclut une vue
  $this->layout->view('ma_vue', $data);
 }
}
```

Ainsi, le $this->news_model nous permet d'avoir accès aux méthodes du modèle. En fait, une fois inclus, le modèle agit exactement comme une bibliothèque.

Renommer le modèle

Si vous avez donné des noms trop longs à vos modèles, vous avez la possibilité de les renommer. Pour cela, il faut utiliser le deuxième paramètre de la méthode model.
`<?php $this->load->model('nom_du_fichier', 'nom_a_donner');`

Voici ce que cela donne avec l'exemple précédent :

Code : PHP - Un contrôleur

```php
<?php

class User extends CI_Controller
{
 public function accueil()
 {
  // Chargement du modèle de gestion des news
  // Nous l'appellerons newsManager
$this->load->model('news_model', 'newsManager');

  $data = array();

  // On lance une requête
$data['user_info'] = $this->newsManager->get_info();

  // Et on inclut une vue
  $this->layout->view('ma_vue', $data);
 }
}
```

Maintenant, nous ne faisons plus
`<?php $this->news_model->get_info();`
mais bien
`<?php $this->newsManager->get_info();`

Vous pourrez penser que c'est un détail mais bien nommer ses modèles permet d'améliorer la lisibilité de votre code. Prenez de bonnes habitudes dès le début.

Nous avons fait le tour de ce qui constitue le modèle. Une fois qu'il est inclus, nous pouvons avoir accès à ses méthodes de la même façon qu'une bibliothèque. Cependant, nous n'avons pas encore eu l'occasion de réaliser de vraies requêtes. La prochaine partie vous permettra de faire vos premiers pas avec la bibliothèque database car nous construirons un modèle pour un système de news.

Le modèle de gestion des news

Notre précédent modèle n'en était pas vraiment un car nous ne lui avons pas fait effectuer de requêtes. Nous nous sommes contentés de lui faire retourner un tableau pour faciliter l'écriture de la méthode. Maintenant, nous allons créer le modèle pour la gestion d'un système de news.

Voici la table :

	Champ	Type	Interclassement	Attributs	Null	Défaut	Extra
☐	id	smallint(5)		UNSIGNED	Non	*Aucun*	auto_increment
☐	auteur	varchar(30)	latin1_swedish_ci		Non	*Aucun*	
☐	titre	varchar(100)	latin1_swedish_ci		Non	*Aucun*	
☐	contenu	text	latin1_swedish_ci		Non	*Aucun*	
☐	date_ajout	datetime			Non	*Aucun*	
☐	date_modif	datetime			Non	*Aucun*	

Le code SQL :

Code : SQL

```sql
CREATE TABLE `news` (
  `id`  SMALLINT(5) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `auteur` VARCHAR(30) NOT NULL,
  `titre`  VARCHAR(100) NOT NULL,
  `contenu` TEXT  NOT NULL,
  `date_ajout` DATETIME NOT NULL,
  `date_modif` DATETIME NOT NULL
)

ENGINE = MyISAM DEFAULT CHARSET = utf8;
```

Et le prototype du modèle :

Code : PHP - application/models/news_models.php

```php
<?php  if ( ! defined('BASEPATH')) exit('No direct script access
allowed');

class News_model extends CI_Model
{
 protected $table = 'news';

 /**
* Ajoute une news
*/
 public function ajouter_news()
 {

 }

 /**
* Édite une news déjà existante
```

```
    */
    public function editer_news()
    {

    }

    /**
    * Supprime une news
    */
    public function supprimer_news()
    {

    }

    /**
    * Retourne le nombre de news
    */
    public function count()
    {

    }

    /**
    * Retourne une liste de news
    */
    public function liste_news()
    {

    }
}


/* End of file news_model.php */
/* Location: ./application/models/news_model.php */
```

Le fait de stocker le nom de la table dans un attribut permet de rendre votre modèle plus flexible.

Le point sur Active Record

CodeIgniter vous propose deux manières de lancer des requêtes à la base de données : en l'écrivant en langage SQL ou en utilisant Active Record.

Active Record est un patron sur lequel on peut s'appuyer pour faire nos requêtes. Il s'utilise en trois étapes :

- création de la requête avec les méthodes de la classe `Database` ;
- exécution de la requête ;
- manipulation des données que la base de données a renvoyées.

Dans la suite de ce tutoriel, j'utiliserai autant que possible Active Record. Maintenant, nous allons implémenter chaque méthode du modèle une par une.

Méthode `ajouter_news`

Cette méthode permet d'ajouter une news. Ainsi, les champs `id`, `date_ajout` et `date_modif` devront être gérés par la méthode. Celle-ci a besoin de 3 paramètres : l'auteur, le titre et le contenu.

Code : PHP - Implémentation de la méthode ajouter_news

```php
<?php

 /**
* Ajoute une news
*
* @param string $auteur L'auteur de la news
* @param string $titre Le titre de la news
* @param string $contenu Le contenu de la news
* @return bool Le résultat de la requête
*/
 public function ajouter_news($auteur, $titre, $contenu)
 {

 }
```

Pour ajouter un nouvel élément dans la base de données, nous pouvons utiliser la méthode `insert`. Nous allons passer le nom de la table au premier paramètre. Ensuite, nous pouvons utiliser le deuxième paramètre pour les données (sous forme d'un tableau associatif) ou bien nous pouvons utiliser la méthode `set` pour remplir champ par champ les données à insérer.

Nous utiliserons les méthodes `set` + `insert` car elles permettent de mieux contrôler l'insertion de données.

Voici le prototype de la méthode `set`.

```php
<?php $this->set ( $nom_du_champ, $valeur, $echappement_automatique = true )
```

Code : PHP - Implémentation de la méthode ajouter_news

```php
<?php

 /**
* Ajoute une news
*
* @param string $auteur L'auteur de la news
* @param string $titre Le titre de la news
* @param string $contenu Le contenu de la news
* @return bool Le résultat de la requête
*/
 public function ajouter_news($auteur, $titre, $contenu)
 {
  // Ces données seront automatiquement échappées
  $this->db->set('auteur',  $auteur);
  $this->db->set('titre',   $titre);
  $this->db->set('contenu', $contenu);

  // Ces données ne seront pas échappées
  $this->db->set('date_ajout', 'NOW()', false);
  $this->db->set('date_modif', 'NOW()', false);

  // Une fois que tous les champs ont bien été définis, on "insert"
le tout
  return $this->db->insert($this->table);
 }
```

Cette technique est particulièrement intéressante car elle nous permet de fixer les valeurs champ par champ. Une fois que nous avons fini, nous utilisons la méthode `insert` en fournissant le nom de la table. Vous remarquerez que le troisième paramètre de la méthode `set` permet d'empêcher l'échappement des données. En effet, la chaîne de caractères `NOW()` ne doit pas être insérée en tant que telle mais elle doit être interprétée.

Vous devrez donc faire très attention aux données que vous insérez lorsque vous fixez ce paramètre à `false`.

Cette méthode pourra être appelée de cette façon depuis un contrôleur :

Code : PHP - Un contrôleur

```php
<?php

public function accueil()
{
 $this->load->model('news_model', 'newsManager');

 $resultat = $this->newsManager->ajouter_news('Arthur',
           'Un super titre',
           'Un super contenu !');
 var_dump($resultat);
}
```

Vous pouvez aussi chaîner votre requête de cette façon :

Code : PHP - Méthode ajouter_news

```php
<?php

public function ajouter_news($auteur, $titre, $contenu)
{
 return $this->db->set('auteur',  $auteur)
   ->set('titre',   $titre)
   ->set('contenu', $contenu)
   ->set('date_ajout', 'NOW()', false)
   ->set('date_modif', 'NOW()', false)
   ->insert($this->table);
}
```

Comme vous le voyez, nous avons réalisé une requête sans utiliser le langage SQL, uniquement avec des méthodes de la classe `Database`.

Méthode modifier_news

Pour cette méthode, nous allons procéder de la même manière qu'avec la méthode `ajouter_news`, sauf que nous n'appellerons pas en dernier la méthode `insert` mais la méthode `update`. De plus, il nous faudra appeler une autre méthode avant `update` afin de renseigner la condition pour mettre à jour le champ, ce sera la méthode `where`.

Comme c'est la première fois que nous modifions des données de la sorte, je commence par le code et je vous expliquerai après.

Code : PHP - Méthode editer_news

```php
<?php

/**
* Édite une news déjà existante
*
* @param integer $id L'id de la news à modifier
* @param string $titre Le titre de la news
* @param string $contenu Le contenu de la news
* @return bool Le résultat de la requête
*/
public function editer_news($id, $titre = null, $contenu = null)
{
 // Il n'y a rien à éditer
 if($titre == null AND $contenu == null)
 {
  return false;
 }
```

```
        // Ces données seront échappées
        if($titre != null)
        {
         $this->db->set('titre', $titre);
        }
        if($contenu != null)
        {
         $this->db->set('contenu', $contenu);
        }

        // Ces données ne seront pas échappées
        $this->db->set('date_modif', 'NOW()', false);

        // La condition
        $this->db->where('id', (int) $id);

        return $this->db->update($this->table);
       }
```

Dans un premier temps, nous mettons à jour les deux champs que nous souhaitons modifier. Si l'un des deux a pour valeur `null`, alors il sera ignoré.
Puis nous complétons la date et l'heure de la mise à jour.
Ensuite, il faut lui dire quelles sont les entrées que nous souhaitons modifier. Ici, nous avons choisi d'autoriser la modification uniquement par le champ `id`. Cela se fait par la méthode `where`. Si nous ne lui disons pas quels sont les champs que nous souhaitons modifier, alors nous modifierons toutes les entrées que possède la table.
Puis en dernier, nous lançons la méthode `update` pour exécuter la requête en lui donnant en paramètre le nom de la table.

Vous pouvez rendre cette méthode plus flexible en autorisant la modification de la news par un autre champ que `id`. En effet, la méthode `where` peut aussi recevoir un tableau associatif du type : `nom_du_champ => valeur`.
Ainsi, vous pouvez créer une condition : « si `$id` est un tableau, alors ce tableau sera donné à la méthode `where` sans spécifier de champ ».

Nous l'implémenterons comme cela :

Code : PHP

```
<?php

// Exemple de valeur pour $id.
$id = 5;
$id = array('id' => 9);
$id = array('pseudo' => 'Arthur',
            'titre' => 'monTitre');

/* ******* */

if(is_array($id))
{
 $this->db->where($id);
}
else
{
 $this->db->where('id', (int) $id);
}
```

En utilisant des méthodes pour générer vos requêtes, ce genre de chose est très facile à faire.

Méthode supprimer_news

Cette méthode n'aura qu'un seul paramètre : l'id de la news à supprimer. Pour la suppression d'entrées, nous utiliserons la méthode `delete`. Nous l'utiliserons une fois la condition `where` définie.

Code : PHP - Méthode supprimer_news

```php
<?php

/**
* Supprime une news
*
* @param integer $id L'id de la news à modifier
* @return bool Le résultat de la requête
*/
public function supprimer_news($id)
{
 return $this->db->where('id', (int) $id)
   ->delete($this->table);
}
```

Méthode count

Pour les méthodes permettant de retourner le nombre d'entrées, la bibliothèque `database` nous propose une méthode toute prête : `count_all_results`.

Pour l'utiliser, il faut dans un premier temps recourir à la méthode `where` pour restreindre le nombre de résultats retournés, puis à la méthode `count_all_results` pour exécuter la requête.

Code : PHP - Méthode count

```php
<?php

/**
* Retourne le nombre de news.
*
* @param array $where Tableau associatif permettant de définir des
conditions
* @return integer Le nombre de news satisfaisant la condition
*/
public function count($where = array())
{
 return (int) $this->db->where($where)
         ->count_all_results($this->table);
}
```

Voici un exemple d'utilisation de cette méthode :

Code : PHP - Un contrôleur

```php
<?php

public function accueil()
{
 $this->load->model('news_model', 'newsManager');

 $nb_news = $this->newsManager->count();
 $nb_news_de_bob = $this->newsManager->count(array('auteur' =>
'Bob'));
}
```

Dans le premier cas, comme nous n'avons pas défini de paramètre, nous avons envoyé à la méthode `where` un tableau vide.

Cela équivaut à retourner le nombre total d'entrées.
Dans le deuxième cas, le tableau que nous donnons en paramètre permet d'affiner la requête et de ne retourner que le nombre d'entrées où le champ « auteur » vaut « Bob ».

Méthode `liste_news`

Les requêtes permettant de sélectionner un ensemble d'entrées fonctionnent de la même manière que les requêtes sous forme de chaînes de caractères.

- Il y a une méthode `select` pour réduire la taille des données récupérées. Sa valeur par défaut est * qui permet de récupérer tous les champs.
- Il y a une méthode `from` qui permet de spécifier la table souhaitée.
- Il y a une méthode `join` pour faire des jointures.
- Il y a des méthodes conditionnelles. Nous avons vu `where`, mais il y en a d'autres (`or_where`, `like`, `or_like`, etc.).
- Il existe aussi d'autres méthodes telles que `limit` ou `order_by`, qui ont les mêmes utilités que dans le langage SQL.

Il en existe beaucoup d'autres mais vous avez remarqué que les noms des méthodes sont les mêmes (ou presque) que les mots-clés du langage SQL.

Intéressons-nous à présent à l'implémentation de la méthode :

Code : PHP - Méthode liste_news

```php
<?php

/**
* Retourne une liste de $nb dernières news.
*
* @param integer $nb Le nombre de news
* @param integer $debut Nombre de news à sauter
* @return objet La liste de news
*/
public function liste_news($nb = 10, $debut = 0)
{
 return $this->db->select('*')
   ->from($this->table)
   ->limit($nb, $debut)
   ->order_by('id', 'desc')
   ->get()
   ->result();
}
```

Cette requête se lit comme cela : « Sélectionne tous les champs de la table $this->table, donne-moi $nb news à partir de la news $debut et ordonne le tout du plus grand id au plus petit.

Attention à l'ordre des paramètres dans la méthode *limit*. Sous CodeIgniter, le premier paramètre correspond au nombre de résultats souhaités et le second est le nombre de résultats à sauter. C'est **le contraire** du langage SQL.

Notre modèle de news

Voici le modèle de news que nous avons réalisé :

Code : PHP - ./application/models/news_model.php

```php
<?php  if ( ! defined('BASEPATH')) exit('No direct script access
allowed');

/**
* News_model
*
* ajouter_news($auteur, $titre, $contenu)
* editer_news($id, $titre = null, $contenu = null)
```

```
 * supprimer_news($id)
 * count($where = array())
 * liste_news($nb = 10, $debut = 0)
 */

class News_model extends CI_Model
{
 protected $table = 'news';

 /**
 * Ajoute une news.
 *
 * @param string $auteur L'auteur de la news
 * @param string $titre Le titre de la news
 * @param string $contenu Le contenu de la news
 * @return bool Le résultat de la requête
 */
 public function ajouter_news($auteur, $titre, $contenu)
 {
  return $this->db->set('auteur', $auteur)
            ->set('titre',  $titre)
    ->set('contenu', $contenu)
    ->set('date_ajout', 'NOW()', false)
    ->set('date_modif', 'NOW()', false)
    ->insert($this->table);
 }

 /**
 * Édite une news déjà existante.
 *
 * @param integer $id L'id de la news à modifier
 * @param string $titre Le titre de la news
 * @param string $contenu Le contenu de la news
 * @return bool Le résultat de la requête
 */
 public function editer_news($id, $titre = null, $contenu = null)
 {
  // Il n'y a rien à éditer
  if($titre == null AND $contenu == null)
  {
   return false;
  }

  // Ces données seront échappées
  if($titre != null)
  {
   $this->db->set('titre', $titre);
  }
  if($contenu != null)
  {
   $this->db->set('contenu', $contenu);
  }

  return $this->db->set('date_modif', 'NOW()', false)
    ->where('id', (int) $id)
    ->update($this->table);
 }

 /**
 * Supprime une news.
 *
 * @param integer $id L'id de la news à modifier
 * @return bool Le résultat de la requête
 */
 public function supprimer_news($id)
 {
  return $this->db->where('id', (int) $id)
    ->delete($this->table);
 }
```

```
        /**
        * Retourne le nombre de news.
        *
        * @param array $where Tableau associatif permettant de définir des
        conditions
        * @return integer Le nombre de news satisfaisant la condition
        */
        public function count($where = array())
        {
         return (int) $this->db->where($where)
                 ->count_all_results($this->table);
        }

        /**
        * Retourne une liste de $nb dernière news.
        *
        * @param integer $nb Le nombre de news
        * @param integer $debut Nombre de news à sauter
        * @return objet La liste de news
        */
        public function liste_news($nb = 10, $debut = 0)
        {
         return $this->db->select('*')
           ->from($this->table)
           ->limit($nb, $debut)
           ->order_by('id', 'desc')
           ->get()
           ->result();
        }
       }


       /* End of file news_model.php */
       /* Location: ./application/models/news_model.php */
```

La réalisation de ce modèle vous permettra de comprendre bien plus facilement la documentation sur la bibliothèque database, et plus particulièrement le pattern Active Record.

Désormais, vous en savez suffisamment sur CodeIgniter pour construire un site web. Vous ne connaissez pas tout de ce framework. Je ne vais pas vous lâcher comme cela, bien entendu. Mais si vous avez bien compris tout ce que nous avons vu dans ces deux parties, vous êtes en mesure de réaliser un bon nombre de sites.

[TP] Un livre d'or

Nous en avons terminé avec la découverte de CodeIgniter : nous allons pouvoir passer aux choses sérieuses. Mais ne vous inquiétez pas, il n'y aura rien de vraiment compliqué, nous allons juste nous servir de tout ce que nous avons appris jusqu'ici

En route !

La lecture des cinq premiers chapitres de cette partie est obligatoire pour bien comprendre ce que nous allons réaliser.

Description du livre d'or

Petit rappel

Un livre d'or est un endroit où les visiteurs inscrivent des remarques sur ce qu'ils pensent de votre site.

Il faudra donc une page permettant d'afficher les commentaires (avec un système de pagination), et une autre où l'on aura un formulaire qui permettra aux visiteurs d'écrire un nouveau commentaire. Un commentaire est composé d'un pseudo, d'un message et d'une date.

Descriptif des pages

Affichage des commentaires

Comme je l'ai dit ci-dessus, cette page sert à afficher les derniers commentaires des visiteurs. J'ai fait une petite capture d'écran de ce que l'on obtient une fois ce TP terminé.

Comme vous le voyez, on a bien les commentaires avec une pagination et un lien vers la page d'ajout d'un commentaire.

Formulaire

La page de formulaire est plutôt simple. Il s'agit de demander au visiteur son pseudo et le message qu'il veut écrire dans le livre d'or. Une fois cela fait, on enregistre le message et on affiche une page de confirmation.

Avant de commencer

Comme vous le voyez, j'ai fait aussi simple que possible. Pas de *captcha*, ni rien de particulièrement difficile que vous ne savez pas faire. Normalement, vous avez tous déjà réalisé un livre d'or ou quelque chose d'équivalent.

Cette partie du tutoriel est uniquement faite pour que vous sachiez ce que nous allons faire. Dans la partie suivante, je vais vous donner des conseils pour vous aider. Vous n'êtes donc pas obligés de la lire. Vous pouvez vous lancer dès maintenant.

Coup de pouce

Comme convenu, cette partie est facultative.

Schéma de la table

Pour vous faire économiser quelques précieuses minutes, je vous ai fait une copie de la requête qui m'a permis de créer ma table. La voici.

Code : SQL - Création de la table « livreor_commentaires »

```
CREATE TABLE IF NOT EXISTS `livreor_commentaires`(

 `id`       int(10)     unsigned NOT NULL AUTO_INCREMENT,
 `pseudo`   varchar(52)     NOT NULL,
 `message`  text            NOT NULL,
 `date`  DATETIME        NOT NULL,

 PRIMARY KEY (`id`)
)

ENGINE = MyISAM DEFAULT CHARSET = utf8 AUTO_INCREMENT = 1;
```

Rien d'exceptionnel ici. On trouve un champ « pseudo », un champ « contenu » et un champ « date ». En image, cela donne ceci :

Sauf que cette table est vide… Pour tester, il vaut mieux la remplir un petit peu. Je vous ai donc concocté un petit ensemble d'entrées à copier-coller dans PHPMyAdmin. Une fois les tests terminés, il vous suffira de vider votre table pour la remettre à zéro.

Secret (cliquez pour afficher)

Code : SQL

```
INSERT INTO `livreor_commentaires` (`id`, `pseudo`, `message`,
`date`) VALUES
(1, 'Chuck Norris', 'bienvenue !', '2010-09-13 21:04:00'),
(2, 'Patrick', 'J''aime !', '2010-09-05 21:04:12'),
(3, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(4, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04 21:05:01'),
(5, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(6, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01 21:05:51'),
(7, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(8, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(9, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(10, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:16'),
(11, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(12, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(13, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(14, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(15, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(16, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(17, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(18, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(19, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:31'),
(20, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(21, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(22, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(23, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(24, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
```

```
21:05:51'),
(25, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(26, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(27, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(28, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:35'),
(29, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(30, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(31, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(32, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(33, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(34, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(35, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(36, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(37, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:36'),
(38, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(39, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(40, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(41, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(42, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(43, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(44, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(45, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(46, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:37'),
(47, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(48, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(49, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(50, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(51, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(52, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(53, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(54, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(55, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:37'),
(56, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(57, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(58, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(59, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(60, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(61, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(62, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(63, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(64, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:38'),
(65, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(66, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(67, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(68, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(69, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(70, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(71, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(72, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(73, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
```

```
21:08:38'),
(74, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(75, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(76, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(77, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(78, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(79, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(80, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(81, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(82, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:39'),
(83, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(84, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(85, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(86, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(87, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(88, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(89, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(90, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(91, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:39'),
(92, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(93, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(94, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(95, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(96, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(97, 'H. Reges', 'Salut, super ce site !', '2010-09-02 21:06:09'),
(98, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(99, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12 21:06:59'),
(100, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:40'),
(101, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(102, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(103, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(104, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(105, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(106, 'H. Reges', 'Salut, super ce site !', '2010-09-02
21:06:09'),
(107, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(108, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12
21:06:59'),
(109, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:40'),
(110, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(111, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(112, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(113, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(114, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(115, 'H. Reges', 'Salut, super ce site !', '2010-09-02
21:06:09'),
(116, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(117, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12
21:06:59'),
(118, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:41'),
```

```
(119, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(120, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(121, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(122, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(123, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(124, 'H. Reges', 'Salut, super ce site !', '2010-09-02
21:06:09'),
(125, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(126, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12
21:06:59'),
(127, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:41'),
(128, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(129, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(130, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(131, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(132, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(133, 'H. Reges', 'Salut, super ce site !', '2010-09-02
21:06:09'),
(134, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(135, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12
21:06:59'),
(136, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:42'),
(137, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13'),
(138, 'Patrick', 'Encore une fois.', '2010-09-02 21:04:45'),
(139, 'Mathieu', 'Ceci est un commentaire.', '2010-09-04
21:05:01'),
(140, 'Francois', 'Il etait une fois....', '2010-09-05 21:05:23'),
(141, 'M. Deres', 'Bienvenue a vous tous !', '2010-09-01
21:05:51'),
(142, 'H. Reges', 'Salut, super ce site !', '2010-09-02
21:06:09'),
(143, 'L. Dergs', 'Il etait une fois...', '2010-09-05 21:06:33'),
(144, 'L. Broue', 'Jamais deux sans toi !', '2010-09-12
21:06:59'),
(145, 'S. Eruit', 'Salut, j''aime bien ce site !', '2010-09-21
21:08:42'),
(146, 'M. Ourit', 'Debut de site sympathique.', '2010-09-13
21:08:13');
```

Voilà, votre table est maintenant prête.

Le système de pagination

Vous n'avez peut-être pas envie de refaire, à partir de rien, tout un système de pagination. Bien pensé (comme celui du SdZ, par exemple), l'algorithme n'est pas évident (comprendre *long à faire*). Par exemple, un système qui évite de faire autant de tours de boucle qu'il existe de pages est intéressant côté performance.

Pour vous faciliter la tâche, vous avez à disposition une bibliothèque native permettant de mettre en place votre pagination très rapidement. C'est celle-ci que j'utiliserai pour la correction de ce TP. Voici un exemple d'utilisation de cette bibliothèque.

Code : PHP

```
<?php

// Chargement de la bibliothèque
$this->load->library('pagination');
```

```
$this->load->library('pagination');

// Initialisation des paramètres d'utilisation de la pagination
define('NB_COMMENTAIRE_PAR_PAGE', 15);
define('NB_COMMENTAIRE_SAUVEGARDE_EN_BDD', 4587);

$this->pagination->initialize(array('base_url' => base_url() .
'index.php/livreor/voir/',
        'total_rows' => NB_COMMENTAIRE_SAUVEGARDE_EN_BDD,
        'per_page' => NB_COMMENTAIRE_PAR_PAGE));

// Récupération du HTML
$html_pagination = $this->pagination->create_links();
```

Votre pagination est donc très rapidement créée avec ce système.

Si la forme ou le contenu de cette pagination ne vous convient pas, il est possible de personnaliser un bon nombre de choses (notamment les délimiteurs). Cependant, si vous êtes comme moi et que vous voulez quelque chose de très spécifique, il faudra réécrire ou surcharger (proprement) cette bibliothèque pour qu'elle se comporte exactement comme vous le souhaitez. Si c'est votre cas, je vous conseille tout de même de conserver le même comportement côté utilisateur, pour que les modifications soient complètement transparentes.

Ce système de pagination n'envoie pas dans l'URL le numéro de la page mais le numéro du commentaire.

Voilà, j'en ai fini pour les coups de pouce. Tout ce qu'il y a dans la correction a été vu dans les cinq précédents chapitres. À vos éditeurs !

Correction

Vous avez terminé ? Ou vous êtes bloqués ? Pas de panique. C'est tout à fait normal d'avoir des difficultés quand on ne connaît pas encore très bien le framework.

Avant de coder, nous allons réfléchir à tout ce qu'il nous faudra pour réaliser ceci.

Analyse et conception

Le contrôleur

Dans un premier temps, il va nous falloir un contrôleur. Celui-ci sera composé du constructeur, qui permettra de charger différents éléments, et de trois méthodes.

- *index* : lorsqu'aucun nom de méthode ne sera spécifié dans l'URL, alors ce sera la méthode *voir* qui sera appelée.
- *voir* : c'est la page qui affiche les commentaires. Elle prendra donc un paramètre facultatif qui sera le numéro de la page, ou du commentaire, selon votre système de pagination.
- *ecrire* : cette page sera celle qui affichera le formulaire pour ajouter un nouveau message.

Le modèle

Une fois cela fait, nous devons définir notre modèle. En effet, nous allons devoir faire des requêtes à la base de données. Pour ce système, nous n'avons besoin que de trois méthodes : une qui renvoie le nombre de messages, une autre qui ajoute un message dans la base de données et la dernière qui récupère une liste de commentaires. J'ai appelé ces méthodes respectivement *ajouter_commentaire*, *count* et *get_commentaires*.

Sous forme de schéma, cela nous donne quelque chose dans ce genre-là :

Les vues

Il ne reste plus qu'à définir nos vues. Nous allons avoir besoin de trois vues.

- *afficher_commentaires* : c'est la vue qui sera appelée pour afficher les commentaires issus de la base de données. Il s'agira notamment d'effectuer une boucle sur le tableau de résultats renvoyé par la requête.
- *ecrire_commentaire* : cette vue comportera le formulaire pour ajouter un commentaire.
- *confirmation* : cette vue ne sera affichée que lorsque le visiteur aura envoyé un commentaire.

Bien, maintenant que nous avons fini d'analyser notre futur livre d'or, nous allons le coder.

Le modèle

Prototype

Commençons par le plus rapide : le modèle. Voici son prototype.

Code : PHP - ./application/models/livreor_model.php

```php
<?php

class Livreor_model extends CI_Model
{
 private $table = 'livreor_commentaires';

 public function ajouter_commentaire($pseudo, $message)
 {

 }

 public function count()
 {

 }

 public function get_commentaires($nb, $debut = 0)
 {

 }
}

/* End of file livreor_model.php */
/* Location: ./application/models/livreor_model.php */
```

Méthode ajouter_commentaire

Cette méthode reçoit deux paramètres, le pseudo et le contenu du message. Avant de lancer la requête qui ajoutera une entrée dans la table, nous allons procéder à une vérification sur ces deux paramètres. Nous vérifierons s'ils sont bien des chaînes de caractères et nous nous assurerons que celles-ci ne sont pas vides.

Code : PHP - implémentation de la méthode ajouter_commentaire

```php
<?php

public function ajouter_commentaire($pseudo, $message)
{
 if(!is_string($pseudo) OR !is_string($message) OR empty($pseudo) OR
empty($message))
```

```
        {
         return false;
        }

        // ...
    }
```

Maintenant, nous pouvons envoyer la requête à la base de données. Nous devons donc appeler la méthode *set* de la bibliothèque *database* puis la méthode *insert* pour lui indiquer que l'on veut ajouter une entrée.

Voici comment cela peut s'implémenter.

Code : PHP - implémentation de la méthode ajouter_commentaire

```
<?php

public function ajouter_commentaire($pseudo, $message)
{
 if(!is_string($pseudo) OR !is_string($message) OR empty($pseudo) OR
empty($message))
  {
   return false;
  }

 return $this->db->set(array('pseudo' => $pseudo,
        'message' => $message))
   ->set('date', 'NOW()', false)
   ->insert($this->table);
}
```

Nous appelons donc une fois la méthode *set* pour les champs « pseudo » et « message » qui seront automatiquement échappés. Puis nous l'appelons une seconde fois pour le champ « date » qui ne doit pas être échappé (ce qui explique la présence du troisième argument). Enfin, nous appelons la méthode *insert* pour lancer la requête.

Méthode count

Celle-ci va être très simple à implémenter car nous disposons de la méthode *count_all* de la bibliothèque *database* qui permet justement de retourner le nombre d'entrées d'une table.

Une autre façon de faire est d'utiliser la méthode *count_all_results*. Celle-ci permet en plus de fixer des contraintes. Mais dans notre cas, nous avons juste besoin qu'elle retourne le nombre d'entrées.

Code : PHP - implémentation de la méthode count

```
<?php

public function count()
{
 return $this->db->count_all($this->table);
}
```

Méthode get_commentaires

Cette méthode attend deux paramètres : le nombre de commentaires que l'on veut récupérer et un entier à partir duquel la requête doit accepter les résultats. Autrement dit, il s'agit en fait des variables que nous allons donner à la méthode *limit*.

Comme la méthode *ajouter_commentaire*, nous allons d'abord vérifier la cohérence des paramètres avant de lancer la requête. Il

s'agira donc de vérifier si les deux variables sont bien des entiers positifs.
Une fois cela fait, nous allons pouvoir lancer la requête et retourner le résultat.

Code : PHP - Implémentation de la méthode get_commentaires

```php
<?php

public function get_commentaires($nb, $debut = 0)
{
 if(!is_integer($nb) OR $nb < 1 OR !is_integer($debut) OR $debut <
0)
 {
  return false;
 }

 return $this->db->select('`id`, `pseudo`, `message`,
DATE_FORMAT(`date`,\'%d/%m/%Y &agrave; %H:%i:%s\') AS \'date\'',
false)
   ->from($this->table)
   ->order_by('id', 'desc')
   ->limit($nb, $debut)
   ->get()
   ->result();
}
```

La méthode *from* est facultative car le premier paramètre de la méthode *get* accepte aussi le nom de la table.

Le deuxième argument de la méthode *select* permet d'enlever l'échappement automatique car nous utilisons la fonction SQL *date_format*. Faites donc attention à ce que vous y mettez.

Nous en avons donc fini avec notre modèle. Voici le code complet.

Secret (cliquez pour afficher)

Code : PHP - ./application/models/livreor_model.php

```php
<?php

class Livreor_model extends CI_Model
{
 private $table = 'livreor_commentaires';

 public function ajouter_commentaire($pseudo, $message)
 {
  if(!is_string($pseudo) OR !is_string($message) OR empty($pseudo)
OR empty($message))
  {
   return false;
  }

  return $this->db->set(array('pseudo' => $pseudo,
         'message' => $message))
    ->set('date', 'NOW()', false)
    ->insert($this->table);
 }

 public function count()
 {
  return $this->db->count_all($this->table);
 }

 public function get_commentaires($nb, $debut = 0)
```

```
    {
     if(!is_integer($nb) OR $nb < 1 OR !is_integer($debut) OR $debut
    < 0)
     {
      return false;
     }

     return $this->db->select('`id`, `pseudo`, `message`,
    DATE_FORMAT(`date`,\'%d/%m/%Y &agrave; %H:%i:%s\') AS \'date\'',
    false)
         ->from($this->table)
         ->order_by('id', 'desc')
         ->limit($nb, $debut)
         ->get()
         ->result();
    }
   }

   /* End of file livreor_model.php */
   /* Location: ./application/models/livreor_model.php */
```

Le contrôleur et les vues

Le prototype du contrôleur

Comme nous l'avons défini dans la partie conception, le contrôleur de notre livre d'or possède quatre méthodes. Il n'y a donc rien de nouveau par rapport au schéma UML.

Code : PHP - Prototype du contrôleur

```
<?php  if ( ! defined('BASEPATH')) exit('No direct script access
allowed');

class Livreor extends CI_Controller
{
 const NB_COMMENTAIRE_PAR_PAGE = 15;

 public function __construct()
 {
  parent::Controller();

  // Chargement des ressources pour tout le contrôleur
  $this->load->database();
  $this->load->helper(array('url', 'assets'));
  $this->load->model('livreor_model', 'livreorManager');
 }

// ------------------------------------------------------------------
-------

 public function index($g_nb_commentaire = 1)
 {
  $this->voir($g_nb_commentaire);
 }

// ------------------------------------------------------------------
-------

 public function voir($g_nb_commentaire = 1)
 {
  // La page qui permet de voir les commentaires.
 }
```

```
// ------------------------------------------------------------------------
-------

 public function ecrire()
 {
  // La page qui permet d'écrire un commentaire.
 }
}


/* End of file livreor.php */
/* Location: ./application/controllers/livreor.php */
```

Il est possible que certaines ressources chargées depuis le contrôleur soient déjà chargées par défaut (via l'autoloader). Si c'est le cas, vous pouvez les enlever si vous voulez (le fait de charger plusieurs fois une même ressource ne pose pas de problème). De plus, notez que nous utiliserons le helper *assets* que nous avons créé au chapitre sur les helpers.

La page des commentaires

Nous allons commencer par réaliser la page qui affichera les commentaires. Nous devons donc, dans cette méthode, récupérer les commentaires dans la base de données et envoyer tout cela à la vue. Pour cela, nous devons vérifier la cohérence de la variable $_GET.
Enfin, nous devons aussi récupérer le code HTML généré par la bibliothèque de pagination.

Voici le code qui permet de faire cela. Je l'ai abondamment documenté. Cela sera donc plus facile pour vous de le comprendre.

Code : PHP - Implémentation de la méthode voir

```
<?php

public function voir($g_nb_commentaire = 1)
{
 $this->load->library('pagination');

 $data = array();

 // Récupération du nombre total de messages sauvegardés dans la
base de données
 $nb_commentaire_total = $this->livreorManager->count();

 // On vérifie la cohérence de la variable $_GET
 if($g_nb_commentaire > 1)
 {
  // La variable $_GET semblent être correcte. On doit maintenant
  // vérifier s'il y a bien assez de commentaires dans la base de
données.
  if($g_nb_commentaire <= $nb_commentaire_total)
  {
   // Il y a assez de commentaires dans la base de données.
   // La variable $_GET est donc cohérente.

   $nb_commentaire = intval($g_nb_commentaire);
  }
  else
  {
   // Il n'y pas assez de messages dans la base de données.

   $nb_commentaire = 1;
  }
 }
 else
 {
  // La variable $ GET "nb commentaire" est erronée. On lui donne
```

```
une
  // valeur par défaut.

  $nb_commentaire = 1;
 }

 // Mise en place de la pagination
 $this->pagination->initialize(array('base_url' => base_url() .
'index.php/livreor/voir/',
         'total_rows' => $nb_commentaire_total,
         'per_page' => self::NB_COMMENTAIRE_PAR_PAGE));

 $data['pagination'] = $this->pagination->create_links();
 $data['nb_commentaires'] = $nb_commentaire_total;

 // Maintenant que l'on connaît le numéro du commentaire, on peut
lancer
 // la requête récupérant les commentaires dans la base de données.
 $data['messages'] = $this->livreorManager-
>get_commentaires(self::NB_COMMENTAIRE_PAR_PAGE, $nb_commentaire-1);

 // On charge la vue
 $this->load->view('livreor/afficher_commentaires', $data);
}
```

Ceci permet donc d'afficher la vue *afficher_commentaires* en lui envoyant les variables $data. Sauf que cette vue n'existe pas. Nous allons corriger cela tout de suite. Voici le contenu de la mienne. Le plus important est en fait le principe de la boucle. Le reste, c'est juste pour faire joli.

Code : PHP - ./application/views/livreor/afficher_commentaires.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >

 <head>
  <title>Un livre d'or avec CodeIgniter</title>
  <meta http-equiv="Content-Type" content="text/html; charset=<?php
echo $this->config->item('charset'); ?>" />
  <link rel="stylesheet" type="text/css" media="screen" href="<?php
echo css_url('livreor/style'); ?>" />
 </head>

 <body>
  <div id="messages">

   <p>
    Il y a actuellement <?php echo $nb_commentaires; ?>
commentaires. <br />
    <?php echo url('écrire un commentaire', 'livreor/ecrire'); ?>
   </p>

   <div class="pagination"><?php echo $pagination; ?></div>

   <?php foreach($messages as $message): ?>
    <div id="num_<?php echo $message->id; ?>" class="message">
     <p>
      <a href="#num_<?php echo $message->id; ?>">#</a>
      Par <span class="pseudo_commentaire"><?php echo
htmlentities($message->pseudo); ?></span>
      le <span class="date_commentaire"><?php echo $message->date; ?
></span>
     </p>
     <div class="contenu_commentaire"><?php echo
nl2br(htmlentities($message->message)); ?></div>
    </div>
```

```
    <?php endforeach; ?>

    <div class="pagination"><?php echo $pagination; ?></div>

  </div>
 </body>

</html>
```

Et voici ma feuille de style CSS. Là encore, c'est juste pour le beauté de la chose.

Code : CSS - ./assets/css/livreor/style.css

```
*
{
 margin: 0;
 padding: 0;
}

.pagination
{
 text-align: center;
}

#messages
{
 margin: auto;
 width: 80%;
}

#messages .message
{
 margin: 1% auto;
 padding: 3%;
 border: 1px outset red;
}

form
{
 margin: 2% auto;
 padding: 1.5%;
 width: 85%;
 border: 1px inset green;
 text-align: center;
}

form input
{
 margin-bottom: 1.5%;
}

form textarea
{
 margin-bottom: 1.5%;
 width: 70%;
}

#confirmation
{
 margin: 1%;
 padding: 1.5%;
 border: 1px outset blue;
 color: rgb(158, 10, 10);
}

.form_erreur
{
```

```
 color: red;
}
```

Maintenant que cela est fait, vous devriez voir la page des commentaires avec la pagination fonctionnelle.

La méthode ecrire

Voici la fin de cette correction : la page permettant d'ajouter un commentaire.

Pour cela, j'ai utilisé la bibliothèque *form_validation* que j'avais décrite dans le chapitre portant sur les bibliothèques. La partie importante de cette méthode se situe surtout sur la mise en place des règles de validation du formulaire. Le reste est plutôt banal.

Comme la méthode précédente, celle-ci est abondamment commentée. Il vous faudra sûrement faire le ménage si vous voulez la conserver dans votre projet.

Code : PHP - Implémentation de la méthode « ecrire »

```
<?php

public function ecrire()
{
 $this->load->library('form_validation');

 // Cette méthode permet de changer les délimiteurs par défaut des
messages d'erreur (<p></p>).
 $this->form_validation->set_error_delimiters('<p
class="form_erreur">', '</p>');

 // Mise en place des règles de validation du formulaire
 // Nombre de caractères : [3,25] pour le pseudo et [3,3000] pour
le commentaire
 // Uniquement des caractères alphanumériques, des tirets et des
underscores pour le pseudo
 $this->form_validation->set_rules('pseudo',  '"Pseudo"',
'trim|required|min_length[3]|max_length[25]|alpha_dash');
 $this->form_validation->set_rules('contenu', '"Contenu"',
'trim|required|min_length[3]|max_length[3000]');

 if($this->form_validation->run())
 {
  // Nous disposons d'un pseudo et d'un commentaire sous une bonne
forme

  // Sauvegarde du commentaire dans la base de données
  $this->livreorManager->ajouter_commentaire($this->input-
>post('pseudo'),
          $this->input->post('contenu'));

  // Affichage de la confirmation
  $this->load->view('livreor/confirmation');
 }
 else
 {
  $this->load->view('livreor/ecrire_commentaire');
 }
}
```

Et avec cela, il vous faut les vues associées. Nous avons donc une vue pour l'affichage du formulaire et une autre pour l'affichage de la confirmation.

Notez que la vue affichant le formulaire affiche aussi les erreurs et affiche à nouveau les valeurs dans les champs en cas d'erreur.

Code : PHP - ./application/views/livreor/ecrire_commentaire.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
 <head>
  <title>Un livre d'or avec CodeIgniter</title>
  <meta http-equiv="Content-Type" content="text/html; charset=<?php
echo $this->config->item('charset'); ?>" />
  <link rel="stylesheet" type="text/css" media="screen" href="<?php
echo css_url('livreor/style'); ?>" />
 </head>
 <body>
  <form method="post" action="">
   <div>
    <label>
     Pseudo :
     <input type="text" name="pseudo" value="<?php echo
set_value('pseudo'); ?>" />
    </label>
    <?php echo form_error('pseudo'); ?>
   </div>
   <div>
    <label>
     Message : <br />
     <textarea name="contenu" rows="7" cols="60"><?php echo
set_value('contenu'); ?></textarea>
    </label>
    <?php echo form_error('contenu'); ?>
   </div>
   <p>
    <input type="submit" value="Valider votre commentaire" />
   </p>
  </form>
 </body>
</html>
```

Et la page de confirmation :

Code : PHP - ./application/views/livreor/confirmation.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
 <head>
  <title>Un livre d'or avec CodeIgniter</title>
  <meta http-equiv="Content-Type" content="text/html; charset=<?php
echo $this->config->item('charset'); ?>" />
  <link rel="stylesheet" type="text/css" media="screen" href="<?php
echo css_url('livreor/style'); ?>" />
 </head>
 <body>
  <div id="confirmation">
   <p>
    Votre message a bien été ajouté à la base de données. Nous vous
remercions de votre commentaire.
   </p>
   <p>
    <?php echo url('Revenir à la page des commentaires', 'livreor');
?>
   </p>
  </div>
 </body>
</html>
```

Cette fois-ci, c'est bon ! Nous avons terminé notre livre d'or. Le visiteur peut voir les commentaires et écrire le sien. Et tout cela avec CodeIgniter.
Et voilà ! Notre livre d'or est fini. Il y a de fortes chances qu'il soit insuffisant pour l'intégrer directement dans votre site, mais vous pouvez toujours l'améliorer.

J'espère que ce TP vous a permis de faire le point avec tout ce que nous avons vu. Je pense qu'il était très important de revoir un peu tout cela, mais en produisant quelque chose d'utile. Cela vous permettra d'aborder la troisième partie un peu plus sereinement.

Ne loupez pas l'introduction de la prochaine partie.

Partie 3 : Utilisations avancées

Cette partie recense des outils et des techniques qu'il est très intéressant de connaître, car ils sont très puissants ou très utiles.

Cependant, si vous venez de finir la deuxième partie, faites une pause. Une simple lecture ne suffira pas pour comprendre correctement les chapitres suivants.

Prenez donc votre temps pour faire des tests, rien ne presse. Il faut absolument que vous maîtrisez la syntaxe de tout ce que l'on a vu dans la deuxième partie, sinon vous ne comprendrez pas ce qui suit.

L'outil Profiler

Le *Profiler* est un outil très important pour les développeurs. Il permet d'afficher très rapidement les performances de la page, les requêtes envoyées à la base de données, les variables $_GET et $_POST ainsi que beaucoup d'autres informations. C'est un outil très précieux pour le développement.

Dans un premier temps, nous verrons comment activer ce module. Ensuite, nous découvrirons les outils à notre disposition pour contrôler les performances de notre application. Enfin, nous modifierons le profiler pour y faire apparaître les sessions.

Activer le profiler

Afficher le profiler

Pour activer le profiler, il vous suffit d'ajouter cette ligne dans votre contrôleur : `<?php $this->output->enable_profiler(true);`

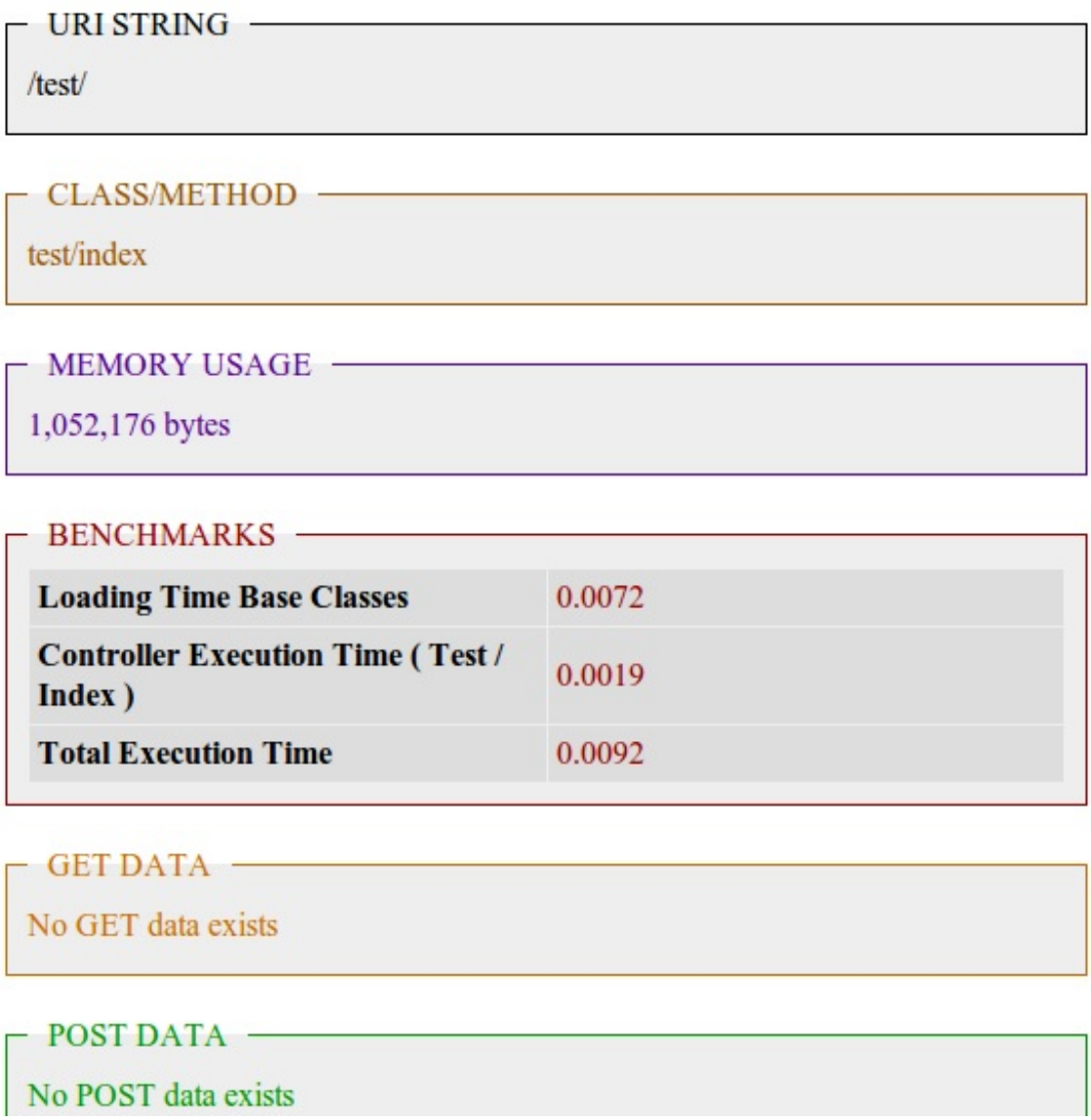
Cette ligne appelle la méthode enable_profiler de la bibliothèque Output.

Exemple : je crée un contrôleur que j'appelle Test et j'y affiche mon profiler.

Code : PHP - ./application/controllers/test.php

```php
<?php

class Test extends CI_Controller
{
 public function index()
 {
$this->output->enable_profiler(true);
 }
}

/* End of file test.php */
/* Location: ./application/controllers/test.php */
```

Celui-ci sera accessible à cette adresse : http://localhost/codeIgniter/index.php/test/. Il vous affichera quelque chose de semblable à ce que montre l'image ci-dessous.



Plutôt pratique, non ?

Réaliser des tests de performance

Vous avez écrit plusieurs requêtes et vous voulez savoir laquelle est la plus performante ? Pas de problème, CodeIgniter a tout prévu !

Il vous suffit d'ajouter une ligne juste avant votre requête et une autre après pour que le temps d'exécution s'affiche dans votre Profiler.

Voici comment faire avec mon exemple précédent :

Code : PHP - ./application/controllers/test.php

```php
<?php

class Test extends CI_Controller
{
 public function index()
 {
   // Première requête
$this->benchmark->mark('requete1_start');
   $query = $this->db->query('SELECT `id`, `username`, `user_rank`
FROM `users`')->result();
$this->benchmark->mark('requete1_end');
```

```
        // Deuxième requête
      $this->benchmark->mark('requete2_start');
        $query = $this->db->select('id, username, user_rank')-
      >from('users')->get()->result();
      $this->benchmark->mark('requete2_end');

        $this->output->enable_profiler(true);
       }
      }

      /* End of file test.php */
      /* Location: ./application/controllers/test.php */
```

Pour que cela fonctionne, vous devez nommer votre test (ici : requete1 ou requete2) et ajouter _start ou _end respectivement avant et après l'objet du test.

J'obtiens maintenant deux lignes supplémentaires dans la catégorie *Benchmarks*.

BENCHMARKS

Loading Time Base Classes	0.0052
Requete1	0.0070
Requete2	0.0207
Controller Execution Time (Test / Index)	0.0369
Total Execution Time	0.0422

Si vous avez besoin de récupérer les résultats du test, vous avez la possibilité d'utiliser cette méthode : `<?php $this->benchmark->elapsed_time('requete1_start', 'requete1_end');`

C'est à peu près tout ce que vous apporte nativement le profiler. C'est très rapide à présenter, mais ça n'enlève rien à son utilité.

Par la suite, nous allons chercher à le modifier.

Modifier l'ordre des différentes catégories

Introduction

Comme je n'allais pas écrire un chapitre dédié uniquement à l'affichage du profiler, j'ai décidé que nous allions modifier son comportement.

Nous n'allons pas faire cela juste pour le plaisir de compliquer la chose ! Ce que nous allons voir maintenant est obligatoire si vous voulez ajouter le contenu des sessions dans le profiler, et d'une manière plus générale, modifier le comportement par défaut d'une bibliothèque de CodeIgniter **sans** toucher au contenu du dossier system.

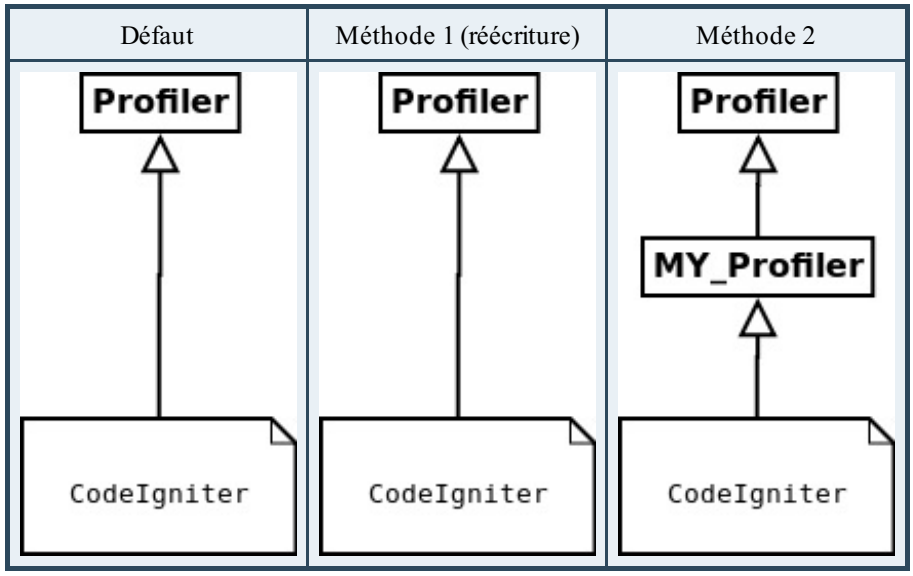
Dans un premier temps, nous changerons l'ordre d'affichage des différents éléments. Par exemple, nous allons montrer en premier les variables $_GET et $_POST et déplacer à la fin la catégorie « mémoire ».

Réécrire les méthodes de CodeIgniter : la théorie

Nous allons faire un petit peu de théorie avant la pratique. Le but n'est pas seulement d'avoir le résultat : il vous faut aussi comprendre comment cela fonctionne pour que vous puissiez faire la même chose pour toutes les bibliothèques.

CodeIgniter a mis en place deux fonctionnalités très simples mais très puissantes pour réécrire les méthodes de ses bibliothèques sans les modifier irrémédiablement.
Nous avons à disposition deux moyens de modifier le comportement de la bibliothèque Profiler.

- **Réécrire entièrement la bibliothèque.** Pour cela, il suffit de créer un fichier Profiler.php dans notre dossier application/libraries/. Les bibliothèques contenues dans le dossier application ont la priorité sur celles du dossier system. Cette technique est particulièrement utile si vous avez beaucoup de méthodes à modifier. Ici, ce ne sera pas le cas.
- **Surcharger certaines méthodes.** Si vous avez bien suivi le cours de POO sur l'héritage, vous devez savoir qu'une classe fille qui hérite d'une classe mère a la possibilité de redéfinir entièrement les méthodes de la classe mère. Nous avons donc besoin de créer une classe qui héritera de la classe Profiler, puis de dire à CodeIgniter d'utiliser notre classe pour appeler les méthodes du profiler. C'est cette méthode que nous allons choisir.



Réécrire les méthodes de CodeIgniter : la pratique

Pour faire cela, vous devez d'abord vous rendre dans votre fichier de configuration (./application/config/config.php) et trouver cette ligne :

Code : PHP - ./application/config/config.php

```php
<?php

/*
|-------------------------------------------------------------------
-------
| Class Extension Prefix
|-------------------------------------------------------------------
-------
|
| This item allows you to set the filename/classname prefix when
extending
| native libraries. For more information please see the user guide:
|
| http://codeigniter.com/user_guide/general/core_classes.html
| http://codeigniter.com/user_guide/general/creating_libraries.html
|
*/
$config['subclass_prefix'] = 'MY_';
```

Si vous lisez la description au-dessus de la déclaration de la variable, vous apprendrez que cette chaîne de caractères permet, dans le dossier libraries, de différencier vos propres bibliothèques de celles servant à surcharger les bibliothèques natives. Autrement dit, en ajoutant le préfixe **MY_** à vos noms de fichiers, CodeIgniter saura que vous voulez réécrire les méthodes de la classe.

Un petit exemple et vous allez comprendre. Créez un fichier appelé MY_Profiler.php dans le dossier ./application/libraries/ et placez-y ce petit bout de code :

Code : PHP - ./application/libraries/MY_Profiler.php

```php
<?php if(!defined('BASEPATH')) exit('No direct script access
allowed');

class MY_Profiler extends CI_Profiler
{

}
```

Si vous rechargez votre page, vous ne devriez voir aucun changement. CodeIgniter utilise maintenant cette classe pour créer son profiler.

Mais pourtant, il n'y a rien dans cette classe…

Faux ! La classe MY_Profiler hérite de Profiler. Ainsi, MY_Profiler possède les mêmes méthodes que Profiler. Pour l'instant, en tout cas.

Le préfixe CI_ devant le nom des classes permet de reconnaître les classes natives de CodeIgniter.

À présent, voyons donc les méthodes que possède la classe CI_Profiler. Voici son prototype :

Code : PHP - ./system/libraries/Profiler.php

```php
<?php

class CI_Profiler {

 var $CI;

  function CI_Profiler();

 // ------------------------------------------------------------------
----

  function _compile_benchmarks();
 function _compile_queries();
 function _compile_get();
 function _compile_post();
 function _compile_uri_string();
 function _compile_controller_info();
 function _compile_memory_usage();

 // ------------------------------------------------------------------
----

 function run();
}
```

Nous pouvons classer les méthodes en trois catégories.

- CI_Profiler est le constructeur (PHP4). Il permet d'initialiser la variable $CI. C'est la valeur de retour de la fonction get_instance. Souvenez-vous, nous l'avons déjà vue lors de la réécriture des fonctions du helper url.
- Chaque méthode qui commence par _compile_ est une catégorie de votre profiler. Chacune de ces méthodes renvoie le code HTML qui sera affiché dans le profiler.
- La méthode run renvoie le code HTML qu'elle collecte auprès de chaque méthode.

Pour modifier l'ordre d'affichage des catégories, nous allons modifier l'ordre dans lequel la méthode run appelle les autres méthodes.

Voici la méthode run telle qu'elle est définie nativement :

Code : PHP - ./system/libraries/Profiler.php

```php
<?php

function run()
{
 $output = "<div id='codeigniter_profiler'
style='clear:both;background-color:#fff;padding:10px;'>";

 $output .= $this->_compile_uri_string();
 $output .= $this->_compile_controller_info();
 $output .= $this->_compile_memory_usage();
 $output .= $this->_compile_benchmarks();
 $output .= $this->_compile_get();
 $output .= $this->_compile_post();
 $output .= $this->_compile_queries();

 $output .= '</div>';

 return $output;
}
```

Pour la modifier, nous devons la réécrire. Notre classe MY_Profiler possédera donc une méthode run grâce à ce code :

Code : PHP - ./application/libraries/MY_Profiler.php

```php
<?php if(!defined('BASEPATH')) exit('No direct script access
allowed');

class MY_Profiler extends CI_Profiler
{
 function run()
 {
  $output = "<div id='codeigniter_profiler'
style='clear:both;background-color:#fff;padding:10px;'>";

  $output .= $this->_compile_uri_string();
$output .= $this->_compile_get();
$output .= $this->_compile_post();
  $output .= $this->_compile_controller_info();
  $output .= $this->_compile_benchmarks();
  $output .= $this->_compile_queries();
$output .= $this->_compile_memory_usage();

  $output .= '</div>';

  return $output;
 }
}
```

Cette méthode run est la même que celle de la classe parente, sauf que l'ordre d'appel des différentes méthodes a été modifié.

Nous avons donc bien modifié une bibliothèque de CodeIgniter proprement, en conservant intact le dossier system.

Ajouter les sessions

Voici maintenant le moment d'ajouter les sessions dans le profiler. Si vous avez bien compris comment nous avons fait pour modifier l'ordre d'affichage des catégories, ajouter les sessions ne posera pas plus de problèmes !

Modification de la méthode run

Pour afficher une catégorie, nous devons l'ajouter dans la méthode run. Sinon, elle ne s'affichera pas.

Code : PHP - application/libraries/MY_Profiler.php

```php
<?php

function run()
{
        $output = "<div id='codeigniter_profiler'
style='clear:both;background-color:#fff;padding:10px;'>";

 $output .= $this->_compile_uri_string();
 $output .= $this->_compile_get();
 $output .= $this->_compile_post();
 $output .= $this->_compile_memory_usage();
 $output .= $this->_compile_benchmarks();
 $output .= $this->_compile_controller_info();
 $output .= $this->_compile_queries();
$output .= $this->_compile_session();

 $output .= '</div>';

 return $output;
}
```

La valeur de retour de la méthode _compile_session s'affichera dans le profiler.

Implémentation de la méthode _compile_session

Patron

Créons d'abord le patron :

Code : PHP - ./application/libraries/MY_Profiler.php

```php
<?php

function _compile_session()
{
 $output  = "\n\n";
 $output .= '<fieldset style="border:1px solid #009999;padding:6px
10px 10px 10px;margin:20px 0 20px 0;background-color:#eee">';
 $output .= "\n";

 $output .= '<legend style="color:#009999;">  ' . 'DONNEES
SESSION' . '  </legend>';
 $output .= "\n";

        $output .= "<div style='color:#009999;font-
weight:normal;padding:4px 0 4px 0'>".'No SESSION data
exists'."</div>";

 return $output . "</fieldset>";
}
```

Il s'agit en fait d'un copié-collé du design des autres catégories.

Tester si les sessions sont activées

Code : PHP - ./application/libraries/MY_Profiler.php

```php
<?php
```

```
function _compile_session()
{
 $output  = "\n\n";
 $output .= '<fieldset style="border:1px solid #009999;padding:6px
10px 10px 10px;margin:20px 0 20px 0;background-color:#eee">';
 $output .= "\n";

 $output .= '<legend style="color:#009999;">  ' . 'DONNEES
SESSION' . '  </legend>';
 $output .= "\n";

if(is_object($this->CI->session))
 {
  // Le contenu de la session

 }
else
 {
  // La session est indéfinie
  $output .= "<div style='color:#009999;font-
weight:normal;padding:4px 0 4px 0'>".'No SESSION data
exists'."</div>";
 }

 return $output . "</fieldset>";
}
```

J'ai choisi d'afficher un message dans le cas où la session n'est pas active. Vous pouvez aussi renvoyer une chaîne de caractères vide pour masquer la catégorie « sessions ». Vous faites comme vous voulez ; c'est votre profiler, après tout. 

SESSION DATA

No SESSION data exists

Réalisation du tableau

Dans cette section, nous afficherons un tableau à deux colonnes : la clé et la valeur associée.

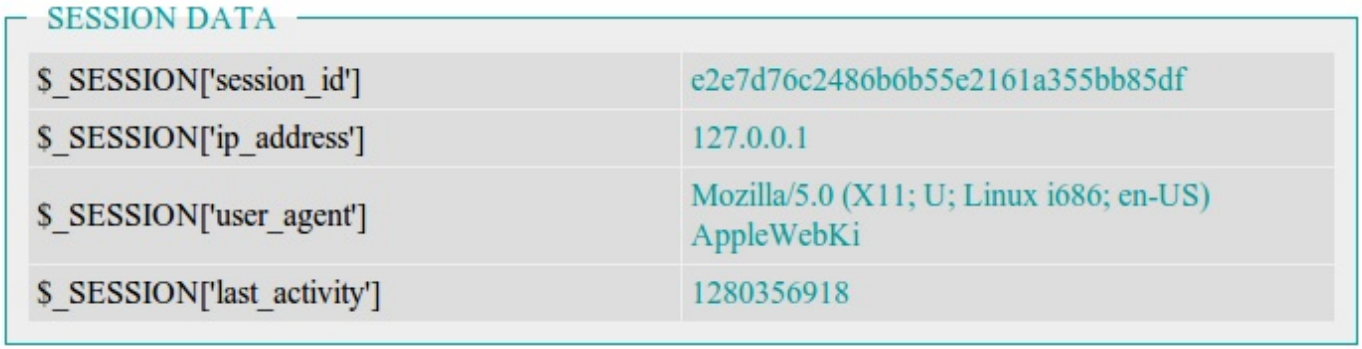
Pour construire ce tableau, nous reprendrons le modèle de la catégorie *Benchmarks*. Je ne m'attarde pas sur le code HTML, c'est juste une histoire de design.

Code : PHP - ./application/libraries/MY_Profiler.php

```
<?php

function _compile_session()
{
 $output  = "\n\n";
 $output .= '<fieldset style="border:1px solid #009999;padding:6px
10px 10px 10px;margin:20px 0 20px 0;background-color:#eee">';
 $output .= "\n";

 $output .= '<legend style="color:#009999;">  ' . 'DONNEES
SESSION' . '  </legend>';
 $output .= "\n";

 if(is_object($this->CI->session))
 {
  // Le contenu de la session
  $output .= "\n\n<table cellpadding='4' cellspacing='1' border='0'
width='100%'>\n";
```

```
        // Nous devons exécuter cette fonction pour récupérer un tableau
      de valeurs
        // En effet, pour accéder aux données de la session, il faut
      récupérer ses attributs
        $sess = get_object_vars($this->CI->session);

        // Nous parcourons chaque valeur du tableau de sessions
        foreach($sess['userdata'] as $key => $val)
        {
         // On échappe (juste pour l'affichage) les données non
      numériques et on les affiche
         if( ! is_numeric($key))
         {
          $key = "'" . $key . "'";
         }
         $output .= "<tr><td width='50%' style='color:#000;
      background-color:#ddd;'>&#36;_SESSION[" . $key . "]  
      </td><td width='50%' style='color:#009999;font-
      weight:normal;background-color:#ddd;'>";

         // On affiche la valeur de la variable. Si c'est un tableau, on
      exécute la fonction print_r
         if(is_array($val))
         {
          $output .= "<pre>" . htmlspecialchars(stripslashes(print_r($val,
      true))) . "</pre>";
         }
         else
         {
          $output .= htmlspecialchars(stripslashes($val));
         }

         $output .= "</td></tr>\n";
        }

        $output .= "</table>\n";
       }
       else
       {
        // La session est indéfinie
        $output .= "<div style='color:#009999;font-
      weight:normal;padding:4px 0 4px 0'>".'No SESSION data
      exists'."</div>";
       }

       return $output . "</fieldset>";
      }
```

Vous voyez maintenant vos sessions dans le profiler. Voici, en image, ce que j'obtiens :



Plutôt sympa, non ?

Modifier le design du profiler

Maintenant que les sessions apparaissent dans votre profiler, vous en avez peut-être ras-le-bol de vous acharner sur la molette de votre souris pour aller voir leur contenu.

Je vous propose donc deux designs alternatifs.

Celui-ci sépare le profiler en deux. Voici le rendu obtenu :

Bien entendu, vous n'êtes pas obligés de faire la même chose, mais l'idée était, je trouve, suffisamment intéressante pour l'évoquer. Je vous donne ma fonction run : adaptez-la à votre résolution si le rendu est de mauvaise qualité.

Code : PHP - ./application/libraries/MY_Profiler.php

```php
<?php

function run()
{
 $output = "<div id='codeigniter_profiler'
style='clear:both;background-color:#fff;padding:10px;'>";

$output .= '<div style="margin-left: 2%; margin-right: 1%; float:
left; width: 47%;">';
 $output .= $this->_compile_uri_string();
 $output .= $this->_compile_controller_info();
 $output .= $this->_compile_memory_usage();
 $output .= $this->_compile_benchmarks();
$output .= '</div>';

$output .= '<div style="margin-left: 51%; margin-right: 2%; width:
auto;">';
 $output .= $this->_compile_get();
 $output .= $this->_compile_post();
 $output .= $this->_compile_queries();
 $output .= $this->_compile_session();
$output .= '</div>';

 $output .= '</div>';

 return $output;
}
```

Celui-ci reprend la même base mais réserve plus de place aux requêtes. Voici le rendu :

En voici le code :

Code : PHP - ./application/libraries/MY_Profiler.php

```php
<?php

function run()
{
 $output = "<div id='codeigniter_profiler'
style='clear:both;background-color:#fff;padding:10px;'>";
```

```
        $output .= '<div style="margin-left: 2%; margin-right: 1%; float:
        left; width: 47%;">';
         $output .= $this->_compile_uri_string();
         $output .= $this->_compile_get();
         $output .= $this->_compile_memory_usage();
        $output .= '</div>';

        $output .= '<div style="margin-left: 51%; margin-right: 2%; width:
        auto;">';
         $output .= $this->_compile_controller_info();
         $output .= $this->_compile_post();
         $output .= $this->_compile_benchmarks();
        $output .= '</div>';

        $output .= '<div style="margin-left: 2%; margin-right: 2%; width:
        auto;">';
         $output .= $this->_compile_queries();
         $output .= $this->_compile_session();
        $output .= '</div>';

                $output .= '</div>';

                return $output;
        }
```

Étant donné que je n'ai absolument aucun talent pour le design, je suis très limité pour fabriquer un joli profiler. Cela dit, rien ne vous empêche de laisser libre cours à votre imagination.
Comme vous le voyez, en une seule instruction, vous avez pu afficher un descriptif très précis sur toutes les pages. C'est un outil qui vous fait gagner énormément de temps pendant le développement. Utilisez-le !

Modification de la classe Model : le CRUD

Vous savez vous servir des modèles, mais vos codes sont redondants ? Alors ce chapitre est fait pour vous ! Nous allons étendre les possibilités de la classe Model pour y intégrer les méthodes CRUD.

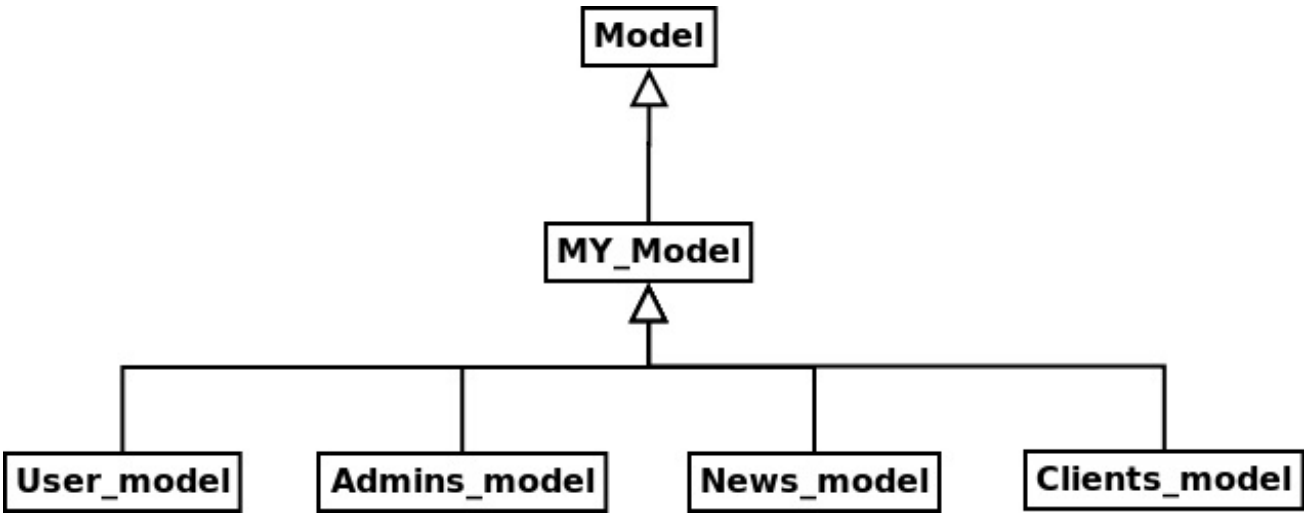
Ainsi, chaque nouveau modèle créé sera équipé de cinq méthodes par défaut.

La théorie

Pour atteindre notre objectif, nous devrons créer une classe qui héritera de la classe Model. Nous l'appellerons MY_Model de façon à ce qu'elle soit inclue automatiquement lorsque nous chargerons la classe Model.

Dans le cas où vous lisez ce tutoriel dans le désordre, le suffixe **MY_** permet de surcharger les fonctionnalités d'une classe. Référez-vous au chapitre concernant le profiler si vous n'avez jamais fait cela.

Voici un schéma de ce que nous allons faire.



Comme vous le voyez, le principe est d'ajouter une classe entre Model et tous les autres modèles que vous allez créer. Comme cela, si vos modèles héritent de MY_Model, vous aurez accès aux méthodes de cette classe-ci sans pour autant créer autre chose qu'un modèle.

Mais alors, nous devrons spécifier à chaque fois le nom de la table pour toutes nos requêtes ?

Non, mes amis ! Souvenez-vous du rôle du modèle et l'environnement dans lequel vous développez. Un modèle sert à proposer un ensemble de méthodes pour **une seule table**. Et nous sommes en programmation orientée objet. On peut donc créer un attribut dans chaque modèle qui sauvegardera le nom de la table auquel ses méthodes se réfèrent.

Exemple : voici deux modèles qui diffèrent uniquement par le nom de la table.

Code : PHP - Modèle de la table Users

```php
<?php

class User_model extends MY_Model
{
protected $table = 'users';
}
```

Code : PHP - Modèle de la table News

```php
<?php

class News_model extends MY_Model
{
```

```
protected $table = 'news';
}
```

Notez bien que les deux classes héritent de My_Model.

Voici comment nous appellerons ces modèles :

Code : PHP - Un contrôleur quelconque

```
<?php

public function index()
{
 // Nous chargeons les deux modèles
 $this->load->model('user_model', 'userManager');
 $this->load->model('news_model', 'newsManager');

 // Nous utiliserons une méthode du CRUD pour chacun des deux
modèles
 $nb_membre = $this->userManager->count();
 $nb_news   = $this->newsManager->count();
}
```

La même méthode retournera donc des valeurs différentes car elle se basera sur l'attribut $table. Cette variable étant différente, il est normal que la requête retourne des valeurs différentes (à moins d'avoir le même nombre de membres que de news).

Création du prototype

Maintenant, nous allons créer le prototype de cette fonction. J'ai volontairement séparé le prototype et l'implémentation car vous n'êtes pas obligés d'écrire les mêmes méthodes que moi. Tout dépend des besoins que vous avez et de la base de données que vous utilisez.

Nous allons donc créer un fichier `MY_Model.php` dans le dossier `./application/core/`.

Voici le prototype :

Code : PHP - ./application/core/MY_Model.php

```
<?php  if ( ! defined('BASEPATH')) exit('No direct script access
allowed');

// ------------------------------------------------------------------------

class MY_Model extends CI_Model
{
 /**
* Insère une nouvelle ligne dans la base de données.
*/
 public function create()
 {

 }

 /**
* Récupère des données dans la base de données.
*/
 public function read()
 {

 }

 /**
```

```
    * Modifie une ou plusieurs lignes dans la base de données.
    */
     public function update()
     {

     }

     /**
    * Supprime une ou plusieurs lignes de la base de données.
    */
     public function delete()
     {

     }

     /**
    * Retourne le nombre de résultats.
    */
     public function count()
     {

     }
    }

    /* End of file MY_Model.php */
    /* Location: ./system/application/core/MY_Model.php */
```

Vous remarquez que j'utilise aussi une méthode `count`, il ne s'agit donc pas d'un CRUD strict.

- Vous pouvez aussi écrire toutes les méthodes que vous voulez, mais elles ne doivent pas être spécifiques à une table en particulier (ou à un modèle, si vous préférez), sinon, vous perdrez tout l'intérêt.
- PyroCMS est un CMS développé avec CodeIgniter et surcharge lui aussi le modèle natif. Mais il va même plus loin et ajoute de nombreuses fonctions. Vous trouverez le fichier en question sur cette page. Regarder le code d'autres développeurs peut parfois être très instructif.

Maintenant que cela est fait, nous pouvons implémenter toutes ces méthodes.

Implémentation des méthodes

Nous allons maintenant remplir les méthodes que nous avons vues.

Pour effectuer nos requêtes, nous utiliserons Active Record de la bibliothèque `database`. Ce sera une occasion de plus de vous y familiariser. Pour rappel, la documentation se trouve à cette adresse (plus exactement ici).

J'utiliserai le modèle suivant pour les tests :

Code : PHP - ./application/models/user_model.php

```
<?php

class User_model extends MY_Model
{
 protected $table = 'users';
}
```

La méthode count

Nous allons commencer par la méthode `count` car c'est la plus compliquée. 

Voici ce que nous voulons pouvoir faire depuis notre contrôleur.

Code : PHP - ./application/controllers/test.php

```php
<?php

public function accueil()
{
 $this->load->model('user_model', 'userManager');

 // Le nombre d'entrées dans la table du modèle userManager
 $nb_membres = $this->userManager->count();

 // Une seule condition
 $nb_messages = $this->userManager->count('pseudo', 'Arthur');

 // Multiples conditions
 $option = array();
 $option['titre']  = 'Mon Super Titre';
 $option['auteur'] = 'Arthur';
 $nb_messages_deux = $this->userManager->count($option);
}
```

La méthode `count` fonctionnera selon les trois cas suivants.

- Absence de paramètre : la méthode renvoie le nombre total d'entrées dans la table.
- Deux chaînes de caractères : le premier est le nom du champ, le second est sa valeur.
- Un tableau associatif en premier paramètre pour indiquer plusieurs conditions.

Pour faire cela, nous utiliserons la méthode `count_all_results`. Mais avant de l'utiliser, il faut sélectionner nos conditions ; vous allez voir, c'est très lisible, comme requête.

Code : PHP - ./application/controllers/test.php

```php
<?php

public function count($champ = array(), $valeur = null) // Si $champ
est un array, la variable $valeur sera ignorée par la méthode
where()
{
 return (int) $this->db->where($champ, $valeur)
                        ->from($this->table)
                        ->count_all_results();
}
```

Dans le cas où vous n'avez pas spécifié de paramètre, la méthode `where` sera ignorée car elle recevra un tableau vide.

Le méthode create

Au tour de la méthode `create`. Cette méthode est plus simple. Voici comment elle fonctionne :

Code : PHP - ./application/controllers/test.php

```php
<?php

public function accueil()
{
 $this->load->model('user_model', 'userManager');

 $options_echappees = array();
 $options_echappees['pseudo'] = 'Arthur';
```

```
 $options_echappees['mot_de_passe'] = 'bonjour';

 $options_non_echappees = array();
 $options_non_echappees['date_inscription'] = 'NOW()';

 // Renvoie false car les paramètres sont vides
 $resultat = $this->userManager->create();

 // Renvoie true sans sauvegarder la date
 $resultat = $this->userManager->create($options_echappees);

 // Renvoie true en sauvegardant la date comme une fonction SQL
 $resultat = $this->userManager->create($options_echappees,
$options_non_echappees);
}
```

Vous remarquez que nous avons deux types de données à insérer : les données échappées et celles non échappées. Pour l'expliquer simplement, j'ai pris l'exemple de la fonction SQL `NOW()`. Si `NOW()` est dans le premier tableau, elle sera considérée comme une chaîne de caractères, mais si elle se trouve dans le second tableau, alors elle sera interprétée. Il faut donc faire très attention au contenu du second tableau.

Je vous donne tout de suite le code de cette méthode :

Code : PHP

```
<?php

public function create($options_echappees = array(),
$options_non_echappees = array())
{
 // Vérification des données à insérer
 if(empty($options_echappees) AND empty($options_non_echappees))
 {
  return false;
 }

 return (bool) $this->db->set($options_echappees)
                        ->set($options_non_echappees, null,
false)
                        ->insert($this->table);
}
```

Comme vous le voyez, le troisième argument de la méthode permet de désactiver l'échappement des caractères. C'est ce que nous faisons avec le tableau `$options_non_echappees`.
Lorsque la méthode `set` reçoit un tableau vide, elle ne fait rien, cela permet donc de simplifier le code au maximum.

La méthode update

La méthode `update` possède trois paramètres. Le premier est la condition. Par exemple, ce pourrait être un tableau associatif tel que celui-ci : `<?php $where = array('pseudo' => 'Arthur', 'mot_de_passe' => 'bonjour');`
Mais dans le cas où c'est un entier qui est donné en paramètre, alors on considérera qu'il s'agit d'un `id`.

Les deux autres paramètres sont les mêmes que pour la méthode `create` : deux tableaux, dont l'un sera échappé et l'autre non.
Voici son code :

Code : PHP - Méthode update

```
<?php

public function update($where, $options_echappees = array(),
$options_non_echappees = array())
```

```
{
 // Vérification des données à mettre à jour
 if(empty($options_echappees) AND empty($options_non_echappees))
 {
  return false;
 }

 // Raccourci dans le cas où on sélectionne l'id
 if(is_integer($where))
 {
  $where = array('id' => $where);
 }

 return (bool) $this->db->set($options_echappees)
                        ->set($options_non_echappees, null,
false)
                        ->where($where)
                        ->update($this->table);

}
```

Le code est très parlant. C'est le même que pour la méthode `create`, mais dans ce cas-ci, puisque nous faisons des modifications, il faut ajouter la méthode `where`, sinon la base de données subira la modification pour toutes les entrées.

La méthode `delete`

Celle-ci est la plus simple. Elle ne prend qu'un seul paramètre : `$where`. Cette variable peut être un tableau associatif ou un entier (il sera alors considéré comme la valeur du champ `id`).

Code : PHP

```
<?php

public function delete($where)
{
 if(is_integer($where))
 {
  $where = array('id' => $where);
 }

 return (bool) $this->db->where($where)
                        ->delete($this->table);
}
```

La méthode `read`

La méthode `read` n'a pas beaucoup d'intérêt car elle est très complexe. Autrement dit, si vous voulez effectuer une requête un tout petit peu plus élaborée qu'une simple lecture de la base de données, vous serez bloqués. Je vous ai quand même implémenté cette méthode, mais on ne gagne pas vraiment de temps en l'utilisant.

Code : PHP

```
<?php
public function read($select = '*', $where = array(), $nb = null,
$debut = null)
{
 return $this->db->select($select)
                 ->from($this->table)
                 ->where($where)
                 ->limit($nb, $debut)
                 ->get()
```

```
                                ->result();
        }
```

Ce chapitre est l'un des plus complexes car il fait appel à des notions compliquées de POO, mais si en plus vous avez des lacunes en SQL, vous aurez beaucoup de difficultés. La classe que propose CodeIgniter pour effectuer des requêtes est très puissante.

Cependant, il faut aussi savoir que vous ne pourrez pas faire toutes vos requêtes avec lui. Parfois, lancer ses requêtes « à l'ancienne » peut s'avérer plus clair.

Code : PHP

```php
<?php

$r = $this->db->query('la requete');
```

Nous venons encore une fois de démontrer que CodeIgniter est un *framework* très extensible. Avec un peu d'inspiration, nous avons réalisé un système très robuste qui supprime toute redondance dans nos modèles.

Mise en place des thèmes

Dans ce chapitre, nous créerons une bibliothèque gérant les différents thèmes de votre site. Dans la plupart des sites, il y a toujours une partie qui ne change pas. Cela se traduit souvent par une inclusion de vue comme ceci :

Code : PHP

```php
<?php

// Les vues qui seront tout le temps inclues
$this->load->view('header.php');
$this->load->view('menu_gauche.php');
$this->load->view('menu_droite.php');

// L'unique partie du site qui change
$this->load->view('ma_vue.php');

// La fermeture des balises ouvertes dans les premières vues
$this->load->view('pied_de_page.php');
```

Nous allons créer une bibliothèque qui nous permettra de gérer tout cela bien plus facilement.

Présentation de l'idée

Nous allons chercher à déléguer tout ce qui concerne l'inclusion de la partie fixe du site à une bibliothèque. C'est cette partie fixe qu'il va falloir créer. Vous devrez donc réunir en un seul fichier votre en-tête, vos menus et vos pieds de page, un peu comme ceci :

Code : PHP - Exemple de layout

```php
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
 <head>

 </head>

 <body>
  <div id="contenu">
   <?php echo $output; ?>
  </div>
 </body>

</html>
```

La variable $output est le contenu changeant de la page. Ce fichier est appelé un *layout* (en français, cela donnerait quelque chose comme « patron »).

Nous souhaitons donc pouvoir charger une ou plusieurs vues, les assembler et montrer le contenu là où nous affichons la variable $output.

Nous allons créer les fichiers nécessaires pour cela. Dans un premier temps, il nous faut la bibliothèque : je l'appellerai Layout. Il nous faudra ainsi un fichier layout.php dans le dossier ./application/libraries/.

Puis il nous faut nos fichiers « layout ». Je les ai placés dans un dossier themes, lui-même situé dans le dossier ./application. Je considère que les layouts sont des composants plutôt fondamentaux, au même titre que les pages d'erreur. Ce ne sont pas vraiment des vues, mais elles contiennent quand même du code HTML. Libre à vous de les placer où vous le souhaitez, bien entendu. Dans ce dossier, j'ai créé un fichier default.php contenant le code ci-dessus.

Avant de commencer à coder, voici le fichier de démarrage de cette bibliothèque :

Code : PHP - ./application/libraries/layout.php

```php
<?php if (!defined('BASEPATH')) exit('No direct script access allowed');

class Layout
{
 private $CI;
 private $output = '';

/*
|===============================================================================
| Constructeur
|===============================================================================
*/

 public function __construct()
 {
  $this->CI = get_instance();
 }

/*
|===============================================================================
| Méthodes pour charger les vues
| . view
| . views
|===============================================================================
*/

 public function view($name, $data = array())
 {

 }

 public function views($name, $data = array())
 {

 }
}

/* End of file layout.php */
/* Location: ./application/libraries/layout.php */
```

Nous possédons deux attributs et trois méthodes.

L'attribut $CI est la valeur de retour de la fonction get_instance. L'attribut $output est le contenu HTML qui sera inséré dans le layout.

Le constructeur nous permettra d'initialiser les attributs. Le deux autres méthodes nous permettront d'inclure des vues, dont le contenu sera ajouté à l'attribut $output.

Implémentation des méthodes view et views

Différence entre les deux méthodes

Nous sommes obligés de créer deux méthodes pour inclure nos vues, car il doit être possible d'inclure plusieurs vues sans inclure autant de fois le layout.

- view permet d'afficher une vue dans un layout.
- views permet de sauvegarder le contenu d'une ou plusieurs vues dans une variable, sans l'afficher immédiatement. Pour l'affichage, il faudra utiliser la méthode view.

Implémentation de la méthode views

Cette méthode recevra le nom de la vue à inclure dans l'attribut $output.

Rappel : le troisième paramètre de la méthode view de la bibliothèque Load permet de retourner le contenu de la vue sans l'afficher.

Code : PHP - Implémentation de la méthode views

```php
<?php

public function views($name, $data = array())
{
 $this->output .= $this->CI->load->view($name, $data, true);
 return $this;
}
```

Retourner l'instance de Layout permet l'exécution de plusieurs méthodes à la chaîne.

Implémentation de la méthode view

La méthode view est un peu plus compliquée. Il faut dans un premier temps ajouter le contenu HTML à l'attribut $output. Mais en plus, il faut inclure le layout juste après.

Code : PHP - Implémentation de la méthode view

```php
<?php

public function view($name, $data = array())
{
 $this->output .= $this->CI->load->view($name, $data, true);

 $this->CI->load->view('../themes/default', array('output' => $this->output));
}
```

Vous pouvez maintenant afficher votre vue à l'intérieur du layout.

Code : PHP - Un contrôleur

```php
<?php

public function accueil()
{
 $this->load->library('layout');

 $this->layout->views('premiere_vue')
       ->views('deuxieme_vue')
       ->view('derniere_vue');
}
```

Cela permet d'éviter la redondance du code.

Sauf… qu'il subsiste un problème. Maintenant, vous n'avez plus accès à tout ce qui se passe dans votre balise <head>. Vous ne pourrez donc plus donner de titre à votre page, ni inclure de fichier CSS, etc.

Nous allons remédier à cela tout de suite.

Les variables du layout

Il n'y a qu'un seul moyen pour transmettre des variables au layout : ajouter des variables dans le tableau au moment où on inclut le layout.

Pour cela, au lieu de créer uniquement un attribut $output, nous allons créer un tableau que nous transmettrons au layout au moment de l'inclusion.

Voici donc la nouvelle version de notre classe Layout :

Code : PHP - ./application/libraries/layout.php

```php
<?php if (!defined('BASEPATH')) exit('No direct script access allowed');

class Layout
{
 private $CI;
private $var = array();

/*
|===============================================================================
| Constructeur
|===============================================================================
*/

 public function __construct()
 {
  $this->CI = get_instance();

$this->var['output'] = '';
 }

/*
|===============================================================================
| Méthodes pour charger les vues
| . view
| . views
|===============================================================================
*/

 public function view($name, $data = array())
 {
$this->var['output'] .= $this->CI->load->view($name, $data, true);

$this->CI->load->view('../themes/default', $this->var);
 }

 public function views($name, $data = array())
 {
$this->var['output'] .= $this->CI->load->view($name, $data, true);
  return $this;
 }
}
```

Maintenant que nous avons défini un tableau en attribut, nous allons pouvoir y ajouter des valeurs à l'aide de méthodes.

Nous allons ajouter les variables $titre et $charset. Ce n'est qu'un exemple, à vous de voir les variables que vous désirez utiliser.

Les deux variables seront initialisées au constructeur. Elles pourront être modifiées par deux méthodes. Le titre par défaut sera composé du nom de la méthode du contrôleur et du nom du contrôleur, séparés par un tiret. Pour cela, nous allons utiliser les attributs de la bibliothèque Router. La variable $charset permettra l'affichage des caractères dans le bon encodage. Elle sera initialisée avec la même valeur que celle définie dans le fichier ./application/config/config.php.

Pour récupérer un élément de la configuration, vous devez utiliser la bibliothèque Config. Pour cela, il existe la méthode item qui prend en paramètre le nom de la clé de configuration.

Voici notre nouveau constructeur :

Code : PHP - Constructeur de ./application/libraries/layout.php

```php
<?php

public function __construct()
{
 $this->CI =& get_instance();

 $this->var['output'] = '';

 // Le titre est composé du nom de la méthode et du nom du
contrôleur
 // La fonction ucfirst permet d'ajouter une majuscule
$this->var['titre'] = ucfirst($this->CI->router->fetch_method()) . '
- ' . ucfirst($this->CI->router->fetch_class());

 // Nous initialisons la variable $charset avec la même valeur que
 // la clé de configuration initialisée dans le fichier config.php
$this->var['charset'] = $this->CI->config->item('charset');
}
```

Et voici mon layout modifié :

Code : PHP - ./application/themes/default.php

```php
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
 <head>
  <title><?php echo $titre; ?></title>
  <meta http-equiv="Content-Type" content="text/html; charset=<?php
echo $charset; ?>" />
 </head>

 <body>
  <div id="contenu">
   <?php echo $output; ?>
  </div>
 </body>

</html>
```

Il nous faut maintenant créer deux méthodes qui nous permettront de modifier ces valeurs.

Code : PHP - Méthodes set_titre et set_charset

```php
<?php

/*
|================================================================================
| Méthodes pour modifier les variables envoyées au layout
| . set_titre
| . set_charset
|================================================================================
*/
public function set_titre($titre)
{
 if(is_string($titre) AND !empty($titre))
 {
  $this->var['titre'] = $titre;
```

```
        return true;
       }
       return false;
      }

      public function set_charset($charset)
      {
       if(is_string($charset) AND !empty($charset))
       {
        $this->var['charset'] = $charset;
        return true;
       }
       return false;
      }
```

Désormais, nous pourrons modifier le titre de la page et son encodage depuis le contrôleur.
Nous allons maintenant aborder l'inclusion des feuilles de style CSS et de code JavaScript.

Ajouter le CSS et le JavaScript

Les méthodes `ajouter_css` et `ajouter_js`

Pour ajouter ces fichiers, nous allons les sauvegarder dans deux tableaux. Ensuite, ce sera au layout de faire une boucle qui parcourra tous les liens que nous avons sauvegardés.

Nous devons donc créer les tableaux dans le constructeur.

Code : PHP - ./application/libraries/layout.php

```
<?php

public function __construct()
{
 $this->CI =& get_instance();

 $this->var['output'] = '';

 // Le titre est composé du nom de la méthode et du nom du
contrôleur
 // La fonction ucfirst permet d'ajouter une majuscule
 $this->var['titre'] = ucfirst($this->CI->router->fetch_method()) .
' - ' . ucfirst($this->CI->router->fetch_class());

 // Nous initialisons la variable $charset avec la même valeur que
 // la clé de configuration initialisée dans le fichier config.php
 $this->var['charset'] = $this->CI->config->item('charset');

$this->var['css'] = array();
$this->var['js'] = array();
}
```

Voici les méthodes (j'ai utilisé les fonctions du helper `assets`) :

Code : PHP - ./application/libraries/layout.php

```
<?php

/*
|================================================================================
| Méthodes pour ajouter des feuilles de CSS et de JavaScript
| . ajouter_css
| . ajouter_js
```

```
    |================================================================================
    */
    public function ajouter_css($nom)
    {
     if(is_string($nom) AND !empty($nom) AND file_exists('./assets/css/' . $nom .
    '.css'))
     {
      $this->var['css'][] = css_url($nom);
      return true;
     }
     return false;
    }

    public function ajouter_js($nom)
    {
     if(is_string($nom) AND !empty($nom) AND file_exists('./assets/javascript/' .
    $nom . '.js'))
     {
      $this->var['js'][] = js_url($nom);
      return true;
     }
     return false;
    }
```

Ces deux méthodes permettront donc d'ajouter, depuis le contrôleur, des feuilles de style ou des scripts JavaScript.

Affichage des liens dans le layout

Nous allons effectuer deux boucles `foreach` pour inclure les liens du tableau CSS et les liens du tableau JavaScript.

Code : PHP - ./application/themes/default.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
 <head>
  <title><?php echo $titre; ?></title>
  <meta http-equiv="Content-Type" content="text/html; charset=<?php
echo $charset; ?>" />
<?php foreach($css as $url): ?>
<link rel="stylesheet" type="text/css" media="screen" href="<?php
echo $url; ?>" />
<?php endforeach; ?>

 </head>

 <body>
  <div id="contenu">
   <?php echo $output; ?>
  </div>

<?php foreach($js as $url): ?>
<script type="text/javascript" src="<?php echo $url; ?>"></script>
<?php endforeach; ?>

 </body>

</html>
```

Notre système de layout est maintenant terminé. Cependant, il est dommage de se limiter à un unique layout : il pourrait être intéressant d'en changer… Allons-y.

Modifier son thème

Vous avez peut-être envie de changer de fichier layout lorsque vous vous trouvez sur un contrôleur spécifique. Avec notre système, c'est actuellement impossible. Pourtant, c'est très simple à mettre en place. Il nous faut :

- un attribut pour stocker le nom du layout que l'on veut charger ;
- une méthode pour modifier le nom de ce layout. Nous en profiterons pour faire une vérification de l'existence du fichier ;
- une modification dans la méthode view : il faut lui dire de prendre en compte le bon layout.

Mise en place de l'attribut

Code : PHP - Mise en place de l'attribut

```php
<?php

private $theme = 'default';
```

Nous créons un simple attribut que nous initialisons avec le thème par défaut. À noter : le thème ne doit pas apparaître dans le tableau de variables car il ne doit pas être utilisé par le layout.

Modification de la méthode view

Pour cette étape, il nous faut remplacer le nom du layout que nous avons fixé par le nom que contient l'attribut $theme.

Code : PHP - Modification de la méthode view

```php
<?php

public function view($name, $data = array())
{
 $this->var['output'] .= $this->CI->load->view($name, $data, true);

$this->CI->load->view('../themes/' . $this->theme, $this->var);
}
```

Mise en place de la méthode set_theme

Cette méthode nous permettra de modifier le thème actuel. Elle ne prendra qu'un seul paramètre : le nom du layout à charger. S'il est incorrect, on ne modifie pas l'attribut.

Code : PHP - Méthode set_theme

```php
<?php

public function set_theme($theme)
{
 if(is_string($theme) AND !empty($theme) AND
file_exists('./application/themes/' . $theme . '.php'))
 {
  $this->theme = $theme;
  return true;
 }
 return false;
}
```

Nous pourrons maintenant modifier très facilement l'habit du site web. Vous pourrez stocker les préférences de l'utilisateur dans un cookie, par exemple. Vous pourrez aussi modifier très facilement le thème de toutes les pages d'un contrôleur en l'indiquant dans son constructeur.

Nous en avons terminé avec cette bibliothèque. N'hésitez pas à ajouter des méthodes spécifiques aux layouts !

Par exemple, dans mon dernier projet, j'ai créé une méthode me permettant d'ajouter rapidement un ou plusieurs *frameworks* JavaScript.
J'ai aussi mis en place, *via* le constructeur, un système qui me permettait d'inclure automatiquement une feuille de style CSS. Son chemin était défini en fonction du nom du contrôleur appelé et de sa méthode.

Ce ne sont que des exemples ; modifiez cette bibliothèque pour qu'elle s'adapte exactement à l'utilisation que souhaitez en faire.

Je ne sais pas si cela peut vous rassurer, mais j'ai réuni dans cette partie un bon nombre de fonctionnalités bien tordues ! Alors si vous lisez ce texte et que vous avez tout compris, félicitations !
Ce tutoriel n'est pas encore terminé ! D'autres parties viendront, soyez patients.

J'espère que CodeIgniter semblera moins mystérieux à tous les Zéros et que l'on trouvera un peu plus de sujets à son propos sur le forum PHP.

N'hésitez pas à donner vos avis concernant ce tutoriel. Si vous avez besoin d'un chapitre que je n'ai pas écrit, ou que vous avez une idée qu'il pourrait être intéressant de développer, n'hésitez pas à m'envoyer un MP en préfixant votre titre d'un [CI].

Remerciements

Je remercie particulièrement gyom pour sa correction et ses idées ainsi que l'équipe des zCorrecteurs qui ont été nombreux à triturer ce tutoriel.