Débuter avec le Zend Framework (approche MVC)

Traduction de l'excellent tutoriel par Rob Allen
: Getting started with the Zend Framework.
par Rob Allen Guillaume Rossolini (Tutoriels Web / SEO / PHP)

Date de publication : avril 2007

Dernière mise à jour : 8 septembre 2007

Ce cours est une introduction très sommaire au Zend Framework, dans le but d'écrire une application **MVC** très simple utilisant une base de données.

I - Introduction

I-A - Préambule

Ce tutoriel a été testé sur la version 1.0.0.RC1 du Zend Framework. Il a de grandes chances de fonctionner sur des versions plus récentes mais pas sur les versions antérieures à 1.0.0.RC1 à cause de la dépendance au helper ViewRenderer.

I-B - Architecture MVC

La méthode traditionnelle pour construire une application PHP est :

```
<?php
include "common-libs.php";
include "config.php";
mysql_connect($hostname, $username, $password);
mysql_select_db($database);
?>
<?php include "header.php"; ?>
<h1>Home Page</h1>
<?php
$sql = "SELECT * FROM news";
$result = mysql_query($sql);
?>
<table>
<?php
while ($row = mysql_fetch_assoc($result))
{
    ?>
    <tr>
    <td><?php echo $row['date_created']; ?></td>
    <td><?php echo $row['title']; ?></td>
    </tr>
    <?php
}
?>
</table>
<?php include "footer.php"; ?>
```

Au long du cycle de vie de l'application, ce type de code devient impossible à maintenir car le client continue de demander des modifications, qui sont codées à plusieurs endroits du code principal.

Une méthode permettant d'améliorer les possibilités de maintenance des applications est de séparer le code en différentes parties (et habituellement en différents scripts) :

Modèle	La partie "modèle" de l'application est celle concernée par les détails des informations à être affichées. Dans l'exemple ci-dessus, c'est le concept de "news". Ainsi, cette partie s'occupe généralement de la "logique d'entreprise" de l'application ; elle a tendance à charger et à sauvegarder vers des bases de données.
Vue	La vue contient les morceaux de l'application qui affichent les informations à l'utilisateur. C'est généralement le HTML.

Contrôleur	Le Contrôleur lie ensemble le Modèle et la Vue pour s'assurer que les informations correctes sont affichées dans la page.

Le Zend Framework utilise l'architecture **Modèle-Vue-Contrôleur** (MVC), utilisée pour faciliter le développement et la maintenance en séparant les composants d'une application.

I-C - Matériel requis

Le Zend Framework a besoin des éléments suivants :

- PHP 5.1.4 (ou ultérieur) ;
- Un serveur Web supportant la fonctionnalité mod_rewrite (ce tutoriel suppose l'utilisation d'Apache).

I-D - Récupérer le framework

Le Zend Framework est disponible à l'adresse **http://framework.zend.com/download/stable** au format .zip ou .tar.gz. Au moment de la rédaction, la version 0.9 est la version actuelle. Vous devez utiliser la version 0.9 pour pouvoir profiter de ce tutoriel.

II - Organisation

II-A - Structure des répertoires

Alors que le Zend Framework n'oblige pas à utiliser une structure particulière de répertoires, la documentation en recommande une. Cette structure suppose que vous ayez un contrôle complet sur la configuration de votre serveur Apache mais, puisque nous voulons nous simplifier la vie, nous allons opérer une légère modification.

Commencez par créer un répertoire "zf-tutorial" dans le dossier racine du serveur Web. Cela signifie que l'URI pour obtenir l'application sera : http://localhost/zf-tutorial/.

Créez la structure suivante pour contenir les fichiers de l'application :

```
zf-tutorial/
 /application
  /controllers
  /models
  /views
   /filters
   /helpers
   /scripts
 /library
 /public
  /images
  /scripts
  /styles
```

Comme vous pouvez le voir, nous avons des dossiers distincts pour les fichiers du Modèle, de la Vue et du Contrôleur de l'application. Les images, scripts (Javascript) et CSS sont situés dans des dossiers distincts, dans le dossier "public". Les fichiers téléchargés du Zend Framework seront placés dans le dossier "library". Si vous utilisez d'autres bibliothèques, vous devriez les y mettre également.

Extrayez l'archive, ZendFramework-0.9.1-Beta.zip dans mon cas, dans un dossier temporaire. Tous les fichiers sont placés dans un sous dossier appelé "ZendFramework-0.9.1-Beta". Copiez le contenu de "ZendFramework-0.9.1-Beta/library/Zend" dans "zf-tutorial/library/". Votre dossier "zf-tutorial/library" devrait maintenant contenir un sous dossier "zend".

II-B - Bootstrapping

II-B-1 - Le concept

Le Contrôleur du Zend Framework, Zend_Controller, est prévu pour supporter des sites avec des URIs propres. Pour y parvenir, toutes les URIs doivent passer par un script unique, index.php, connu en tant que "". Cela nous fournit un point central pour toutes les pages de l'application et nous assure que l'environnement est correctement mis en place pour exécuter l'application. Nous y parvenons au moyen d'un fichier .htaccess dans le répertoire "zf-tutorial" :

zf-tutorial/.htaccess

```
RewriteEngine on
RewriteRule .* index.php

php_flag magic_quotes_gpc off
php_flag register_globals off
```

La commande RewriteRule est vraiment simple et peu être interprêtée comme "pour toute URI, utilise index.php".

Nous pouvons également mettre en place quelques paramètres php.ini pour la sécurité. Ils devraient déjà être corrects mais nous voulons en être certains !

La directive php_flag ne fonctionne que si vous utilisez mod_php. Si vous utilisez fast-CGI, alors vous devez vérifier que votre php.ini est correct.

Cependant, les images, scripts et CSS ne doivent pas passer par index.php : en les plaçant dans "public", nous pouvons aisément configurer Apache au moyen d'un autre .htaccess afin d'envoyer directement ces fichiers :

zf-tutorial/public/.htaccess
```
RewriteEngine off
```

Bien que cela ne soit pas strictement nécessaire sans règles de réécriture locales, nous pouvons ajouter quelques .htaccess supplémentaires pour nous assurer que les répertoires "application" et "library" sont protégés :

zf-tutorial/application/.htaccess
```
deny from all
```

zf-tutorial/library/.htaccess
```
deny from all
```

Pour que les .htaccess fonctionnent avec Apache, il faut que la directive de configuration ***AllowOverride*** *soit mise à* ***All*** *dans votre fichier* ***httpd.conf****. L'idée des multiples fichiers .htaccess vient de l'article de Jayson Minard "****Blueprint for PHP Applications: Bootstrapping (part 2)****". Je vous recommande de lire les deux cours.*

II-B-2 - Le script : index.php

Notre script est "**zf-tutorial/index.php**" et nous allons commencer avec le code suivant :

zf-tutorial/index.php
```
<?php
error_reporting(E_ALL|E_STRICT);
date_default_timezone_set('Europe/London');
set_include_path('.'
    . PATH_SEPARATOR . './library'
    . PATH_SEPARATOR . './application/models/'
    . PATH_SEPARATOR . get_include_path());

include "Zend/Loader.php";
Zend_Loader::loadClass('Zend_Controller_Front');

// setup controller
$frontController = Zend_Controller_Front::getInstance();
$frontController->throwExceptions(true);
$frontController->setControllerDirectory('./application/controllers');

// run!
$frontController->dispatch();
```

*Nous ne mettons pas de ?> à la fin du script puisque ce n'est pas nécessaire et que cela peut donner lieu à des erreurs difficiles à identifier en cas d'utilisation de la fonction **header()**, en cas d'espaces additionnels après cette balise.*

Étudions maintenant ce script.

```
error_reporting(E_ALL|E_STRICT);
date_default_timezone_set('Europe/Paris');
```

Si **display_errors=on** dans votre php.ini, alors ces lignes vous assurent que vous verrez les erreurs à l'écran. Nous précisons également notre zone temporelle, tel qu'il est requis depuis PHP 5.1+ : évidemment, il faut mettre ici votre propre zone.

```
set_include_path('.'
    . PATH_SEPARATOR . './library'
    . PATH_SEPARATOR . './application/models/'
    . PATH_SEPARATOR . get_include_path());
include "Zend/Loader.php";
```

Le Zend Framework est constitué tel que ses scripts doivent être dans l'*include path* de PHP. Nous y mettons également les Modèles afin de pouvoir charger plus facilement nos classes par la suite. Pour démarrer, nous avons besoin du script Zend/Loader.php pour nous donner accès à la classe Zend_Loader, qui dispose des méthodes statiques nécessaires au chargement de toute autre classe du Zend Framework.

```
Zend_Loader::loadClass('Zend_Controller_Front');
```

Zend_Loader::loadClass charge la classe en paramètre. Le procédé est de remplacer les caractères "_" du nom de la classe par des sépérateurs de répertoire, puis d'ajouter ".php". Ainsi, la classe "Zend_Controller_Front" est chargée depuis le script "Zend/Controller/Front.php". Si vous suivez la même convention pour vos classes, vous pourrez également utiliser la méthode **Zend_Loader::loadClass()** pour les charger. La première classe dont nous avons besoin est le contrôleur primaire.

Le contrôleur primaire est une classe de routage pour lier l'URI demandée et la fonction PHP correcte permettant d'afficher la page. Afin que le routeur puisse agir, il doit découvrir la portion de l'URI qui désigne l'index.php afin de pouvoir déterminer quels sont les éléments de l'URI à partir de là. C'est effectué par un objet Request, qui s'arrange très bien pour deviner l'URI fondamentale. Cependant, si cela ne fonctionne pas dans votre environneent, vous pouvez la surcharger avec la méthode **$frontController->setBaseUrl()**.

Nous devons configurer le contrôleur primaire afin de trouver où trouver les contrôleurs :

```
$frontController = Zend_Controller_Front::getInstance();
$frontController->setControllerDirectory('./application/controllers');
$frontController->throwExceptions(true);
```

Puisque ceci est un tutoriel, nous utilisons un système de test : j'ai donc décidé de demander au contrôleur primaire de lancer toutes les exceptions qui peuvent survenir. Par défaut, le contrôleur primaire les attrape toutes à notre place et les enregistre dans la propriété "_exceptions" de l'objet Response créé. Cet objet contient toutes les informations sur la réponse à l'URI demandée. Le contrôleur primaire va automatiquement envoyer les en têtes et afficher le contenu de la page juste avant de terminer son travail.

Cela peut être assez déroutant pour les développeurs qui découvrent le framework, ainsi il est plus facile pour vous de relancer de manière à rendre plus visibles les exceptions. Bien entendu, en environnement de production, il ne faudrait de toute manière pas afficher les erreurs à l'utilisateur !

Nous arrivons finalement au c#ur de la question et nous exécutons l'application :

```
// run!
$frontController->dispatch();
```

Si vous allez à **http://localhost/zf-tutorial/** pour essayer, vous devriez voir quelque chose de similaire à :

```
Fatal error: Uncaught exception 'Zend_Controller_Dispatcher_Exception' with message 'Invalid
 controller specified (index)' in...
```

Cela nous informe que nous n'avons pas encore mis en place notre application. Avant de pouvoir le faire, nous devrions étudier ce que nous allons programmer, concentrons-nous donc là-dessus dès à présent.

II-C - Le site Web

II-C-1 - Thème du site

Nous allons construire une liste très simple de notre collection de CDs. La page principale va nous permettre d'afficher la liste et d'ajouter, de modifier ou de supprimer des disques. Nous allons enregistrer notre liste dans une base de données au schéma suivant :

Champ	Type	NULL ?	Commentaires
id	Integer	Non	Clef primaire, auto incrémentation
artist	Varchar(100)	Non	
title	Varchar(100)	Non	

II-C-2 - Pages requises

Les pages suivantes seront nécessaires :

- Page d'accueil : Cela affichera la liste des disques, fournira des liens pour les modifier et supprimer ainsi que pour en ajouter ;
- Ajouter un album : Cette page proposera un formulaire permettant d'ajouter un disque ;
- Modifier un album : Cette page proposera un formulaire permettant de modifier un disque ;
- Supprimer un album : Cette page confirmera que nous souhaitons supprimer un album, puis le supprimera.

II-C-3 - Organiser les pages

Avant de mettre en place les scripts, il faut comprendre comment le framework s'attend à ce que les pages soient organisées. Chaque page de l'application est connue comme une "action" et les actions sont regroupées en "contrôleurs". Par exemple pour une URI du format "http://localhost/zf-tutorial/actualités/voir", le contrôleur est "actualités" et l'action est "voir". Cela permet de regrouper les actions en relation. Par exemple, un contrôleur "actualités" peut avoir les actions "récentes", "archives" et "voir". Le système MVC du Zend Framework supporte également le regroupement de contrôleurs mais notre application n'est pas suffisamment conséquente pour qu'il soit nécessaire de s'en préoccuper !

Le contrôleur du Zend Framework réserve une action "index" comme action par défaut. C'est-à-dire que pour l'URI "http://localhost/zf-tutorial/actualités/", l'action "index" est exécutée. Le framework réserve également un nom de contrôleur si aucun n'est fourni dans l'URI : aucune surprise qu'il soit également appelé "index". Ainsi, l'URI "http://localhost/zf-tutorial/" appelle le contrôleur "index" avec l'action "index".

Puisque c'est un cours simple, nous n'allons pas nous compliquer avec des choses "complexes" comme une séquence de connexion ! Cela attendra un prochain tutoriel...

Puisque nous avons quatre actions à appliquer à tous les albums, nous allons les regrouper en un seul contrôleur comme quatre actions. Nous utiliserons le contrôleur par défaut et les actions sont :

Page	Contrôleur	Action
Accueil	index	index
Ajouter un album	index	ajouter
Modifier un album	index	modifier
Supprimer un album	index	supprimer

Simple, non ?

III - Le Contrôleur

III-A - Mise en place du Contrôleur

Nous sommes maintenant prêts à mettre en place le contrôleur. Dans le Zend Framework, le contrôleur est une classe qui doit être appelée "**{Nom du contrôleur}Controller**".

{Nom du contrôleur} *doit commencer par une lettre majuscule.*

Cette classe doit être dans un script appelé **{Nom du contrôleur}Controller.php** dans le répertoire du contrôleur spécifié. De nouveau, {Nom du contrôleur} doit commencer par une lettre majuscule et ne contenir que des minuscules par la suite. Chaque action est une fonction publique dans le contrôleur et doit être appelée **{nom de l'action}Action**. Dans ce cas, **{nom de l'action}** doit commencer par une lettre minuscule.

Notre contrôleur est donc nommé **IndexController** et défini dans "**zf-tutorial/application/controllers/IndexController.php**" :

zf-tutorial/application/controllers/IndexController.php

```
<?php

class IndexController extends Zend_Controller_Action
{
    function indexAction()
    {
    }

    function ajouterAction()
    {
    }

    function modifierAction()
    {
    }

    function supprimerAction()
    {
    }
}
```

Initialement, nous l'avons défini afin que chaque action affiche son nom. Essayez cela en allant aux adresses suivantes :

URI	Texte affiché
http://localhost/zf-tutorial/	dans IndexController::indexAction()
http://localhost/zf-tutorial/index/ajouter	dans IndexController::ajouterAction()
http://localhost/zf-tutorial/index/modifier	dans IndexController::modifierAction()
http://localhost/zf-tutorial/index/supprimer	dans IndexController::supprimerAction()

Note du traducteur : J'ai traduit les noms des actions afin d'obtenir des URIs propres et en français, mais on voit facilement que les méthodes portent des noms bien malheureux. On a par exemple l'impression de vouloir "supprimer une action" alors qu'il s'agit de "l'action supprimer".

Nous avons maintenant mis en place les quatre actions que nous souhaitons utiliser. Elles ne fonctionneront cependant pas avant que nous ayons mis en place les Vues.

IV - La Vue (les gabarits)

IV-A - Mise en place de la Vue

Le composant Vue du Zend Framework, sans surprise, est nommé **Zend_View**. Ce composant nous aidera à séparer le code d'affichage du code des méthodes d'action.

L'usage fondamental de Zend_View est :
```
$view = new Zend_View();
$view->setScriptPath('/path/to/view_files');
echo $view->render('view.php');
```

Il est évident que si nous devions utiliser ce squelette dans chacune de nos méthodes d'action, nous serions en train de répéter le code de préparation qui n'a pas d'intérêt pour l'action. Nous devrions plutôt initialiser la Vue autre part, puis accéder depuis chaque méthode d'action à notre objet déjà créé.

Les concepteurs du Zend Framework ont prévu ce type de problème, ainsi une solution est prévue pour nous dans un "action helper" (assistant d'action). **Zend_Controller_Action_Helper_ViewRenderer** créé une propriété de vue ($this->view) pour que nous puissions l'utiliser et rend également un script de vue. Pour le rendu, l'assistant demande à l'objet Zend_View de regarder dans **views/scripts/{nom du contrôleur}** afin de rechercher les scripts de Vue à rendre. Le rendement d'un tel script est réalisé par render(), qui va rendre (par défaut, du moins) le script "{controller name}.phtml" et le concaténer au corps de la réponse de l'objet Response. Cet objet est utilisé pour rassembler les en têtes, le corps et les exceptions générées comme résultat de l'utilisation du système MVC. Le contrôleur primaire envoie les headers suivi du contenu du corps à la fin de la répartition (*dispatch*).

Pour intégrer la Vue à notre application, nous devons initialiser la Vue avec la méthode init(). Nous devons également créer des scripts de Vue avec du code de test d'affichage.

Voici les modifications à IndexController :

zf-tutorial/application/controllers/IndexController.php
```
<?php

class IndexController extends Zend_Controller_Action
{
    function indexAction()
    {
        $this->view->title = "Mes albums";
    }

    function ajouterAction()
    {
        $this->view->title = "Ajouter un nouvel album";
    }

    function modifierAction()
    {
        $this->view->title = "Modifier un album";
    }

    function supprimerAction()
    {
        $this->view->title = "Supprimer un album";
    }
}
```

Dans chaque méthode, nous assignons une propriété $title et c'est tout !

L'affichage effectif n'est pas fait pour le moment, car il est pris en charge par le contrôleur primaire à la fin de la répartition.

Nous devons maintenant ajouter nos quatre scripts d'action à notre application. Ces scripts sont connus comme des "gabarits" (*templates*) et la méthode render() s'attend à ce que chaque gabarit s'appelle en fonction de son action et qu'il porte l'extension ".phtml" pour montrer que c'est un gabarit. Le script doit être dans un sous dossier dont le nom dépend du contrôleur, ainsi les quatre scripts sont :

zf-tutorial/application/views/scripts/index/index.phtml
```
<html>
<head>
 <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
 <h1><?php echo $this->escape($this->title); ?></h1>
</body>
</html>
```

zf-tutorial/application/views/scripts/index/ajouter.phtml
```
<html>
<head>
 <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
 <h1><?php echo $this->escape($this->title); ?></h1>
</body>
</html>
```

zf-tutorial/application/views/scripts/index/modifier.phtml
```
<html>
<head>
 <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
 <h1><?php echo $this->escape($this->title); ?></h1>
</body>
</html>
```

zf-tutorial/application/views/scripts/index/supprimer.phtml
```
<html>
<head>
 <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
 <h1><?php echo $this->escape($this->title); ?></h1>
</body>
</html>
```

Tester les quatre actions devrait afficher les titres en gras.

IV-B - Code HTML en commun

Il devient très vite évident que nous avons beaucoup de code HTML répété dans nos Vues. Nous allons mettre en facteur dans le répertoire "scripts" le code qui est commun à deux vues : "header.phtml" et "footer.phtml".

Les nouveaux fichiers sont :

zf-tutorial/application/views/scripts/header.phtml
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
 <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
<div id="content">
```

zf-tutorial/application/views/scripts/footer.phtml
```
</div>
</body>
</html>
```

De nouveau, notre Vue a besoin d'être modifiée :

zf-tutorial/application/views/scripts/index/index.phtml
```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('footer.phtml'); ?>
```

zf-tutorial/application/views/scripts/index/ajouter.phtml
```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('footer.phtml'); ?>
```

zf-tutorial/application/views/scripts/index/modifer.phtml
```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('footer.phtml'); ?>
```

zf-tutorial/application/views/scripts/index/supprimer.phtml
```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('footer.phtml'); ?>
```

IV-C - Ajout de styles

Bien qu'il s'agisse d'un tutoriel simple, nous aurons besoin d'un fichier CSS pour que notre application paraisse un minimum présentable ! Cela pose en fait un problème mineur, car nous ne savons pas réellement comment référencer la CSS puisque l'URI n'indique pas le bon répertoire racine. Pour y remédier, nous utilisons la méthode **getBaseUrl()** qui est dans la requête et nous l'envoyons à la Vue. Cela nous donne la partie de l'URI que nous ne connaissons pas.

Nous utilisons la méthode IndexController::init() puisqu'init() est appelée depuis le constructeur, ce qui rend la propriété accessible dans toutes les actions :

zf-tutorial/application/controllers/IndexController.php
```
...
class IndexController extends Zend_Controller_Action
{
    function init()
    {
        $this->view->baseUrl = $this->_request->getBaseUrl();
    }
```

zf-tutorial/application/controllers/IndexController.php

```
    function indexAction()
    {
...
```

Nous devons ajouter la CSS à la section <head> du fichier header.phtml :

zf-tutorial/application/views/scripts/header.phtml

```
...
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
 <title><?php echo $this->escape($this->title); ?></title>
 <link rel="stylesheet" type="text/css" media="screen"
             href="<?php echo $this->baseUrl;?>/public/styles/site.css" />
</head>
...
```

Enfin, le style :

zf-tutorial/public/styles/site.css

```
body,html {
 font-size:100%;
 margin: 0;
 font-family: Verdana,Arial,Helvetica,sans-serif;
 color: #000;
 background-color: #fff;
}

h1 {
 font-size:1.4em;
 color: #800000;
 background-color: transparent;
}

#content {
 width: 770px;
 margin: 0 auto;
}

label {
 width: 100px;
 display: block;
 float: left;
}

#formbutton {
 margin-left: 100px;
}

a {
 color: #800000;
}
```

Cela devrait donner un rendu un peu meilleur !

V - Le Modèle (la base de données)

V-A - Introduction

Maintenant que nous avons séparé le contrôle de l'application de la Vue affichée, il est temps de passer à la partie Modèle de l'application. Rappelez-vous que le Modèle est la partie du MVC qui s'occupe de l'objectif central de l'application (la *logique applicative*) et ainsi, dans notre cas, dialogue avec la base de données. Nous utiliserons la classe **Zend_Db_Table** qui recherche, ajoute, modifie ou supprime des enregistrements de la base de données.

V-B - Configuration

Pour utiliser Zend_Db_Table, nous avons besoin de lui dire quelle base de données utiliser, ainsi que le nom d'utilisateur et le mot de passe. Puisque nous préférons ne pas inclure ces informations dans le code de l'application, nous allons utiliser un fichier de configuration pour les conserver.

Le Zend Framework propose une classe Zend_Config qui fournit un accès orienté objet aux fichiers de configuration. Ce fichier peut être soit un fichier INI soit XML. Nous utiliserons la méthode INI :

zf-tutorial/application/config.ini
```
[general]
db.adapter = PDO_MYSQL
db.config.host = localhost
db.config.username = rob
db.config.password = 123456
db.config.dbname = zftest
```

Pensez évidemment à mettre vos propres informations, pas les miennes !

L'utilisation de Zend_Config est vraiment facile :

```
$config = new Zend_Config_Ini('config.ini', 'section');
```

Dans le cas ci-dessus, Zend_Config_Ini charge une section depuis le fichier INI, pas toutes les sections (bien que ce soit possible si on le souhaite). Zend_Config_Ini lit également le "point" dans le paramètre comme un séparateur de répertoires pour regrouper les paramètres en relation les uns avec les autres. Dans notre fichier config.ini, l'hôte, le nom d'utilisateur, le mot de passe et le nom de la base de données seront regroupés dans ***$config->db->config****.*

Nous allons charger notre fichier de configuration dans notre Contrôleur (index.php) :

zf-tutorial/index.php
```
...
Zend_Loader::loadClass('Zend_Controller_Front');
Zend_Loader::loadClass('Zend_Config_Ini');
Zend_Loader::loadClass('Zend_Registry');

// load configuration
$config = new Zend_Config_Ini('./application/config.ini', 'general');
$registry = Zend_Registry::getInstance();
$registry->set('config', $config);
```

zf-tutorial/index.php
```
// setup controller
...
```

Nous chargeons les classes que nous allons utiliser (Zend_Config_Ini et Zend_Registry) puis nous chargeons la section "general" de "application/config.ini" dans l'objet $config. Enfin, nous assignons l'objet $config au registre afin de pouvoir le réutiliser ailleurs dans l'application.

Nous n'avons pas réellement besoin d'ajouter $config au registre dans ce tutoriel, mais c'est néanmoins une bonne pratique dans la mesure où dans une application "réelle" vous aurez probablement davantage d'informations que les données d'accès à la base de données. De plus, soyez vigilant car le registre est un peu comme une globale et il cause des dépendances, si vous n'y prenez pas garde, entre des objets qui ne devraient pas dépendre les uns des autres.

V-C - Mise en place de Zend_Db_Table

Pour utiliser Zend_Db_Table, nous devons lui indiquer la configuration que nous venons juste de récupérer. Pour cela, nous devons créer une instance de Zend_Db et l'enregistrer avec la méthode statique **Zend_Db_Table::setDefaultAdapter()**. À nouveau, nous effectuons cette opération dans le bootstrapper (index.php) :

zf-tutorial/index.php
```
...
Zend_Loader::loadClass('Zend_Controller_Front');
Zend_Loader::loadClass('Zend_Config_Ini');
Zend_Loader::loadClass('Zend_Registry');
Zend_Loader::loadClass('Zend_Db');
Zend_Loader::loadClass('Zend_Db_Table');

// load configuration
$config = new Zend_Config_Ini('./application/config.ini', 'general');
$registry = Zend_Registry::getInstance();
$registry->set('config', $config);

// setup database
$db = Zend_Db::factory($config->db->adapter,  $config->db->config->toArray());
Zend_Db_Table::setDefaultAdapter($db);

// setup controller
...
```

V-D - Créer la table

J'utilise MySQL et la requête SQL est la suivante :

```
CREATE TABLE album (
    id int(11) NOT NULL auto_increment,
    artist varchar(100) NOT NULL,
    title varchar(100) NOT NULL,
    PRIMARY KEY (id)
)
```

Exécutez cette requête avec un client comme MySQL ou le client standard en lignes de commandes.

V-E - Ajouter des enregistrements

Nous allons également ajouter quelques enregistrements de manière à pouvoir tester la fonctionnalité de récupération de la page d'accueil :

```
INSERT INTO album (artist, title)
VALUES
('James Morrison', 'Undiscovered'),
('Snow Patrol', 'Eyes Open');
```

V-F - Mise en place du Modèle

Zend_Db_Table est une classe abstraite, nous devons donc en hériter pour définir notre classe spécifique aux albums. Peu importe comment nous appelons notre classe mais il est logique de lui donner le nom de la table de la base de données. Ainsi, notre classe s'appelle "Album" pour correspondre à notre table "album". Pour indiquer à Zend_Db_Table le nom de la table qu'il devra gérer, nous devons remplir la propriété *protected* $_name avec le nom de la table. De plus, Zend_Db_Table suppose que votre table a une clef primaire appelée "id" qui est auto incrementée par la base de données. Le nom de ce champ peut être précisé également, si besoin est.

Nous allons enregistrer la classe Album dans le répertoire "models" :

zf-tutorial/application/models/Album.php

```
<?php

class Album extends Zend_Db_Table
{
    protected $_name = 'album';
}
```

Pas très compliqué, n'est-ce pas ? Heureusement pour nous, nos besoins sont très simples et Zend_Db_Table dispose déjà de toutes les fonctionnalités dont nous avons besoin. Cependant, si vous avez besoin de comportements spécifiques pour gérer votre Modèle, c'est dans cette classe que cela se passe. En général, les fonctions supplémentaires que vous pourrez vouloir implémenter sont des méthodes de type "recherche" (*find*) afin de permettre la sélection des informations dont vous avez besoin. Vous pouvez également indiquer à Zend_Db_Table les noms des tables liées, ainsi l'objet récupèrera également les données liées.

V-G - Afficher les albums

Maintenant que nous avons mis en place la configuration et les informations de la base de données, nous pouvons entrer dans le vif du sujet et afficher quelques albums. C'est dans l'IndexController que cela se passe.

Il est clair que chaque action dans IndexController permettra de manipuler la table "album" à l'aide de la classe "Album", ainsi il est logique de charger la classe d'album lorsque le Contrôleur est instancié. Cela se fait au moyen de la méthode init() :

zf-tutorial/application/controllers /IndexController.php

```
...
    function init()
    {
        $this->view->baseUrl = $this->_request->getBaseUrl();
        Zend_Loader::loadClass('Album');
    }
```

zf-tutorial/application/controllers /IndexController.php
```
...
```

C'est un exemple utilisant Zend_Loader::loadClass() pour charger nos propres classes car nous avons placé le répertoire "models" dans l'include path de PHP (cf. index.php).

Nous allons afficher une liste des albums dans indexAction() :

zf-tutorial/application/controllers/IndexController.php
```
...
function indexAction()
{
    $this->view->title = "Mes albums";
    $album = new Album();
    $this->view->albums = $album->fetchAll();
}
...
```

La méthode **Zend_Db_Table::fetchAll()** retourne un **Zend_Db_Table_Rowset** qui nous permet de parcourir les tuples retournés dans le script de gabarit :

zf-tutorial/application/views/scripts/index/index.phtml
```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<p><a href="<?php echo $this->baseUrl; ?>/index/ajouter">Ajouter un nouvel album</a></p>
<table>
<tr>
 <th>Title</th>
 <th>Artist</th>
 <th> </th>
</tr>

<?php foreach($this->albums as $album) : ?>
<tr>
 <td><?php echo $this->escape($album->title);?></td>
 <td><?php echo $this->escape($album->artist);?></td>
 <td>
   <a href="<?php echo $this->baseUrl; ?>/index/modifier/id/<?php
           echo $album->id;?>">Modifier</a>
  <a href="<?php echo $this->baseUrl; ?>/index/supprimer/id/<?php
      echo $album->id;?>">Supprimer</a>
 </td>
</tr>
<?php endforeach; ?>
</table>
<?php echo $this->render('footer.phtml'); ?>
```

http://localhost/zf-tutorial/ devrait maintenant afficher une jolie liste de (deux) albums.

V-H - Ajouter des albums

Nous allons maintenant programmer une fonctionnalité qui ajoute de nouveaux albums.

V-H-1 - Récupérer le formulaire soumis et enregistrement en base de données

Cela se fait dans ajouterAction() :

zf-tutorial/application/controllers/IndexController.php

```
...
    function ajouterAction()
    {
        $album = new Album();
        $this->view->title = "Ajouter un nouvel album";

        if ($this->_request->isPost()) {
            Zend_Loader::loadClass('Zend_Filter_StripTags');
            $filter = new Zend_Filter_StripTags();

            $artist = $filter->filter($this->_request->getPost('artist'));
            $artist = trim($artist);
            $title = trim($filter->filter($this->_request->getPost('title')));

            if ($artist != '' && $title != '') {
                $data = array(
               'artist' => $artist,
               'title'  => $title,
           );
           $album = new Album();
           $album->insert($data);

                $this->_redirect('/');
                return;
            }
        }

        // set up an "empty" album
        $this->view->album = $album->createRow();;

        // additional view fields required by form
        $this->view->action = 'ajouter';
        $this->view->buttonText = 'Ajouter';
    }
...
```

*Nous utilisons la méthode **$this->_request->isPost()** pour vérifier que le formulaire a été soumis. Si c'est bien le cas, nous récupérons l'artiste et le titre en utilisant le tableau POST et la classe **Zend_Filter_StripTags** pour nous assurer qu'aucun HTML n'est transmis. Ensuite, en supposant qu'ils ont été remplis, nous utilisons notre classe Modèle Album pour ajouter les informations dans un nouvel enregistrement de la base de données.*

Après avoir ajouté l'album, nous redirigeons en utilisant la méthode **_redirect()** du Contrôleur afin de revenir à la racine de l'application.

Enfin, nous allons mettre en place la Vue pour le formulaire que nous utiliserons dans le gabarit. En regardant plus loin, nous pouvons voir que le formulaire de modification sera très similaire à celui-ci, ainsi nous allons utiliser un fichier de gabarit identique (_form.phtml) qui sera appelé depuis ajouter.phtml comme modifier.phtml.

V-H-2 - Afficher un formulaire permettant de soumettre un album

Voici les gabarits :

zf-tutorial/application/views/scripts/index/ajouter.phtml

```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('index/_form.phtml'); ?>
<?php echo $this->render('footer.phtml'); ?>
```

zf-tutorial/application/views/scripts/index/_form.phtml

```
<form action="<?php echo $this->baseUrl ?>/index/<?php
    echo $this->action; ?>" method="post">
<div>
    <label for="artist">Artiste</label>
    <input type="text" name="artist"
        value="<?php echo $this->escape(trim($this->album->artist));?>"/>
</div>
<div>
    <label for="title">Titre</label>
    <input type="text" name="title"
        value="<?php echo $this->escape($this->album->title);?>"/>
</div>

<div id="formbutton">
<input type="hidden" name="id" value="<?php echo $this->album->id; ?>" />
<input type="submit" name="add"
    value="<?php echo $this->escape($this->buttonText); ?>" />
</div>
</form>
```

C'est du code relativement simple. Puisque nous avons l'intention d'utiliser form.phtml également pour le formulaire de modification, nous avons utilisé une variable vers $this->action plutôt que de mettre l'attribut directement dans le code. De la même manière, nous utilisons une variable pour le texte à afficher dans le bouton d'envoi.

V-I - Modifier un album

Modifier un album est presque identique à en ajouter un, donc le code se ressemble :

zf-tutorial/application/controllers/IndexController.php

```
...
    function modifierAction()
    {
        $this->view->title = "Modifier l'album";
        $album = new Album();

        if ($this->_request->isPost()) {
            Zend_Loader::loadClass('Zend_Filter_StripTags');
            $filter = new Zend_Filter_StripTags();

            $id = (int)$this->_request->getPost('id');
            $artist = $filter->filter($this->_request->getPost('artist'));
            $artist = trim($artist);
            $title = trim($filter->filter($this->_request->getPost('title')));

            if ($id !== false) {
                if ($artist != '' && $title != '') {
                    $data = array(
                        'artist' => $artist,
                        'title'  => $title,
                    );
                    $where = 'id = ' . $id;
                    $album->update($data, $where);

                    $this->_redirect('/');
                    return;
                } else {
                    $this->view->album = $album->fetchRow('id='.$id);
                }
            }
        } else {
            // album id should be $params['id']
```

zf-tutorial/application/controllers/IndexController.php
```
            $id = (int)$this->_request->getParam('id', 0);
            if ($id > 0) {
                $this->view->album = $album->fetchRow('id='.$id);
            }
        }

        // additional view fields required by form
        $this->view->action = 'modifier';
        $this->view->buttonText = 'Mettre à jour';
    }
...
```

Lorsque nous sommes en mode "POST", nous récupérons le paramètre "id" depuis les paramètres de l'URI à l'aide de la méthode getParam().

Voici le gabarit :

zf-tutorial/application/views/scripts/index/modifier.phtml
```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('index/_form.phtml'); ?>
<?php echo $this->render('footer.phtml'); ?>
```

Il ne vous aura pas échappé que les méthodes ajouterAction() et modifierAction() sont très similaires. Il faut mettre en facteur ici aussi.

Je vous ai laissé le soin de le faire (comme un exercice), cher lecteur...

V-J - Supprimer un album

Pour terminer notre application, nous devons ajouter la suppression. Nous avons un lien "supprimer" à côté de chaque album de notre liste et l'approche naïve serait de supprimer un album lorsque le lien est utilisé. Ce serait une erreur. Souvenez-vous des spécifications : il ne faut pas faire d'action irréversible en utilisant GET mais plutôt POST. L'accélérateur bêta de Google a rappelé cette leçon à de nombreuses personnes.

Nous allons afficher un formulaire de confirmation à l'utilisateur et, s'il clique "oui", nous effectuerons la suppression.

Le code ressemble quelque peu à celui des sections d'ajout et de modification :

zf-tutorial/application/controllers/IndexController.php
```
...
    function supprimerAction()
    {
        $this->view->title = "Supprimer l'album";

        $album = new Album();
        if ($this->_request->isPost()) {
            Zend_Loader::loadClass('Zend_Filter_Alpha');
            $filter = new Zend_Filter_Alpha();

            $id = (int)$this->_request->getPost('id');
            $del = $filter->filter($this->_request->getPost('del'));

            if ($del == 'Oui' && $id > 0) {
                $where = 'id = ' . $id;
                $rows_affected = $album->delete($where);
            }
```

zf-tutorial/application/controllers/IndexController.php

```
            } else {
                $id = (int)$this->_request->getParam('id');
                if ($id > 0) {
                    // only render if we have an id and can find the album.
                    $this->view->album = $album->fetchRow('id='.$id);

                    if ($this->view->album->id > 0) {
                        // render template automatically
                        return;
                    }
                }
            }

            // redirect back to the album list unless we have rendered the view
            $this->_redirect('/');
        }
    ...
```

De nouveau, nous utilisons cette même astuce (vérifier la méthode d'envoi) pour savoir si nous devons afficher le formulaire ou bien effectuer la suppression, à l'aide de la classe Album. Tout comme l'ajout et la modification, la suppression est faite au moyen d'un appel à **Zend_Db_Table::delete()**.

Nous revenons immédiatement après remplir le corps de la réponse. De cette manière, nous redirigeons vers la liste des albums à la fin de la fonction. Ainsi, au cas où l'une des diverses vérification de validité échoue, nous revenons à la liste des albums sans devoir appeler _redirect() à de multiples reprises dans la méthode.

Le gabarit est un simple formulaire :

zf-tutorial/application/views/scripts/index/supprimer.phtml

```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php if ($this->album) :?>
<form action="<?php echo $this->baseUrl ?>/index/supprimer" method="post">
<p>&Ecirc;tes-vous certain de vouloir supprimer
  '<?php echo $this->escape($this->album->title); ?>' par
  '<?php echo $this->escape($this->album->artist); ?>'?
</p>
<div>
  <input type="hidden" name="id" value="<?php echo $this->album->id; ?>" />
  <input type="submit" name="del" value="Oui" />
  <input type="submit" name="del" value="Non" />
</div>
</form>
<?php else: ?>
<p>Ne peut pas trouver l'album.</p>
<?php endif;?>
<?php echo $this->render('footer.phtml'); ?>
```

VI - Conclusion

VI-A - Résolution de problèmes

Si vous avez des problèmes pour utiliser toute autre action qu'*index*, alors le plus probable est que le routeur est dans l'impossibilité de déterminer dans quel répertoire se trouve votre site Web. Selon mes tentatives jusqu'ici, cela survient lorsque l'URI vers votre site est différente du chemin du répertoire à partir de la racine du site.

Si le code par défaut ne fonctionne pas pour vous, alors vous devriez modifier la $baseURL à la valeur correcte pour votre serveur :

zf-tutorial/index.php
```
...
// setup controller
$frontController = Zend_Controller_Front::getInstance();
$frontController->throwExceptions(true);
$frontController->setBaseUrl('/mysubdir/zf-tutorial/');
$frontController->setControllerDirectory('./application/controllers');
...
```

Vous devrez remplacer "/mysubdir/zf-tutorial/" par le chemin correct vers index.php. Par exemple, si votre URL vers index.php est "http://localhost/~ralle/zf_tutorial/index.php", alors la valeur correcte de $baseURL est "/~ralle/zf_tutorial/".

VI-B - Épilogue

Cela met un terme à notre brève mais complètement fonctionnelle application MVC utilisant le Zend Framework. J'espère que vous avez apprécié et que vous avez appris quelque chose. Si vous trouvez une erreur, veuillez m'envoyer un e-mail à **rob@akrabat.com** !

Ce cours a mis l'accent sur les fonctionnalités les plus simples du framework ; il y a bien plus de classes à explorer en détail ! Vous devriez réellement lire **le manuel** et regarder **le wiki** pour plus d'informations ! Si le développement du projet vous intéresse, alors **le wiki de développement** vaut le coup d'oeil...

VI-C - Liens

Ressources Developpez :

- Tutoriel d'**initiation au Zend Framework**, par Julien Pauli ;
- **Forum d'entraide au Zend Framework**.

Ressources externes :

- Le tutoriel original : **Getting started with the Zend Framework**, par Rob Allen ;
- **Un site dédié au Zend Framework**.