


# Débuter avec Zend Framework 1.5 (approche MVC)






par Rob Allen (auteur) Guillaume Rossolini (traducteur) ([Tutoriels Web / SEO / PHP](#))

Date de publication : avril 2007

Dernière mise à jour : 1 janvier 2010

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.


Traduction de l'excellent tutoriel par Rob Allen :  **Getting started with the Zend Framework** (tutoriel version 1.5.2).

 *Cet article est prévu pour la version 1.5 de Zend Framework, il n'explique pas les améliorations apportées à ce framework dans les versions plus récentes. Pour les versions ultérieures, merci de vous reporter à la traduction plus récente disponible ici :  **Débuter avec Zend Framework (approche MVC), tutoriel pour ZF 1.8 et 1.9** (article original ar Rob Allen, traduction par Sylvain Jorge Do Marco).*

I - Introduction.....	3
I-A - Préambule.....	3
I-B - Architecture MVC.....	3
I-C - Matériel requis.....	4
I-D - Pré suppositions.....	4
I-E - Récupérer le framework.....	4
II - Organisation.....	4
II-A - Structure des répertoires.....	4
II-B - Bootstrapping.....	5
II-B-1 - Le concept.....	5
II-B-2 - Le script : index.php.....	6
II-C - Le site Web.....	7
II-C-1 - Thème du site.....	7
II-C-2 - Pages requises.....	8
II-C-3 - Organiser les pages.....	8
III - Le Contrôleur.....	8
III-A - Mise en place du Contrôleur.....	8
IV - La Vue (les gabarits).....	9
IV-A - Mise en place de la Vue.....	9
IV-B - Code HTML en commun.....	11
IV-C - Ajout de styles.....	12
V - Le Modèle (la base de données).....	14
V-A - Introduction.....	14
V-B - Configuration.....	14
V-C - Mise en place de Zend_Db_Table.....	15
V-D - Créer la table.....	15
V-E - Ajouter des enregistrements.....	15
V-F - Mise en place du Modèle.....	16
V-G - Afficher les albums.....	16
V-H - Ajouter des albums.....	17
V-I - Modifier un album.....	19
V-J - Supprimer un album.....	21
VI - Conclusion.....	22
VI-A - Résolution de problèmes.....	22
VI-B - Épilogue.....	22
VI-C - Liens.....	22

## I - Introduction

### I-A - Préambule

 Ce tutoriel a été testé sur la version 1.5.0 du Zend Framework. Il a de grandes chances de fonctionner sur des versions plus récentes mais pas sur les versions antérieures à 1.5.0. Si vous avez des erreurs 404 en essayant d'atteindre toute autre page que la page d'accueil, veuillez vous assurer que vous avez mis **AllowOverride All** dans la configuration d'Apache.

### 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) :

<p><b>Modèle</b></p>	<p>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</p>
----------------------	--

	tendance à charger et à sauvegarder vers des bases de données.
<b>Vue</b>	La vue contient les morceaux de l'application qui affichent les informations à l'utilisateur. C'est généralement le HTML.
<b>Contrôleur</b>	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 - Pré suppositions

Je suppose que vous utilisez PHP 5.1.4 ou ultérieur, ainsi qu'un serveur Web **Apache**. Votre installation Apache doit avoir l'extension **mod\_rewrite** (installée et configurée).

 *Vous devez également vous assurer qu'Apache est configuré pour accepter les fichiers **.htaccess**. Cela se fait habituellement en modifiant la configuration `AllowOverride None` à **AllowOverride All** dans votre fichier `httpd.conf`.*

Vérifiez les détails exacts dans la documentation de votre distribution. Vous ne pourrez naviguer sur aucune autre page que la page d'accueil si vous n'avez pas convenablement configuré `mod_rewrite` et l'utilisation de `.htaccess`.

## I-E - Récupérer le framework

Le Zend Framework est disponible à l'adresse <http://framework.zend.com/download> au format `.zip` ou `.tar.gz`.

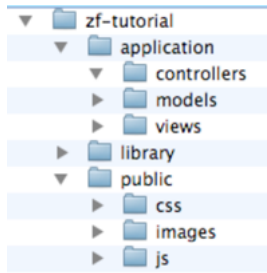
## II - Organisation

### 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 "tutoriel-zf" dans le dossier racine du serveur Web. Cela signifie que l'URI pour obtenir l'application sera : `http://localhost/tutoriel-zf/`.

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



Comme vous pouvez le voir, nous avons des dossiers distincts pour les fichiers du Modèle, de la Vue et du Contrôleur de l'application. Le répertoire "public/" est la racine du site, ce qui signifie que l'URL pour voir le site sera : `http://tutoriel-zf/public/`. C'est prévu pour que la majorité des fichiers de l'application ne soient pas accessibles directement par Apache et soient ainsi plus à l'abri.

**i** Dans un site réel, vous feriez un **VirtualHost** pour le site et vous attribueriez directement le répertoire "public" à la racine (document root). Vous pourriez par exemple créer un **VirtualHost** `tutoriel-zf.localhost` avec ces paramètres :

```
<VirtualHost *:80>
  ServerName tutorial-zf.localhost
  DocumentRoot /var/www/html/tutorial-zf/public
  <Directory "/www/cs">
    AllowOverride All
  </Directory>
</VirtualHost>
```


Le site serait alors accessible par l'adresse `http://tutoriel-zf.localhost/` (assurez-vous que votre fichier `/etc/hosts` ou `c:\windows\system32\drivers\etc\hosts` renvoie bien `tutoriel-zf.localhost` vers 127.0.0.1).

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-1.5.0.zip` dans mon cas, dans un dossier temporaire. Tous les fichiers sont placés dans un sous dossier appelé "ZendFramework-1.5.0". Copiez le contenu de "ZendFramework-0.9.1-Beta/library/Zend" dans "tutoriel-zf/library". Votre dossier "tutoriel-zf/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`. Cette approche est connue en tant que  **design pattern** Front Controller. 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 "tutoriel-zf" :

```
tutoriel-zf/.htaccess
# Règles de réécriture pour Zend Framework
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule .* index.php


# Sécurité : Ne pas autoriser le parcours de répertoires
Options -Indexes
```


### tutoriel-zf/.htaccess

```
# Configuration PHP
php_flag magic_quotes_gpc off
php_flag register_globals off
php_flag short_open_tag on
```

La commande **RewriteRule** est vraiment simple et peut être interprétée comme "pour toute URI qui ne correspond pas à un fichier existant sur le disque, utiliser index.php à la place".

Nous mettons également en place quelques paramètres **php.ini** pour la sécurité et la cohérence des données, ainsi que **short\_open\_tag** à "on" pour utilisation dans les scripts de vue. 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 **CGI/FastCGI**, alors vous devez vérifier que votre **php.ini** est correct.

 Pour pouvoir utiliser les fichiers **.htaccess**, la directive **AllowOverride** doit avoir la valeur **On** dans votre fichier **httpd.conf**.

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

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


### tutoriel-zf/index.php

```
<?php
error_reporting(E_ALL|E_STRICT);
ini_set('display_errors', 1);
date_default_timezone_set('Europe/Paris');

// mise en place des répertoires et chargement des classes
set_include_path('.')
    . PATH_SEPARATOR . './library'
    . PATH_SEPARATOR . './application/models/'
    . PATH_SEPARATOR . get_include_path();
include "Zend/Loader.php";
Zend_Loader::registerAutoload();

// 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 puisque 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);
ini_set('display_errors', 1);
date_default_timezone_set('Europe/Paris');
```

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.

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 permet alors d'appeler sa méthode **registerAutoload()**. Cet appel permet à PHP d'aller chercher automatiquement les classes au moment de leur utilisation.

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

Nous devons configurer le contrôleur primaire afin de lui indiquer 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 route vers un contrôleur **ErrorController** pour nous. 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 !

Le **FrontController** utilise une classe de routage pour faire correspondre l'URL demandée à la bonne méthode nécessaire à l'affichage de la page. Pour que le routeur puisse fonctionner, il doit connaître l'URL jusqu'au script **index.php** afin de pouvoir déduire les éléments qui suivent. C'est l'objet **Request** qui s'en occupe. Il s'en charge habituellement très bien tout seul mais, si cela ne fonctionne pas pour votre configuration, vous pouvez utiliser la fonction **\$frontController->setBaseUrl()** pour forcer la valeur.

Nous arrivons finalement au cœur de la question et nous exécutons l'application :

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

Si vous allez à <http://localhost/tutoriel-zf/> 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 Zend 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/tutoriel-zf/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 de Zend Framework réserve une action "index" comme action par défaut. C'est-à-dire que pour l'URI "http://localhost/tutoriel-zf/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/tutoriel-zf/" 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. Avec 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 **"tutoriel-zf/application/controllers/IndexController.php"** :

```
tutoriel-zf/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
<a href="http://localhost/tutoriel-zf/public">http://localhost/tutoriel-zf/public</a>	dans IndexController::indexAction()
<a href="http://localhost/tutoriel-zf/public/index/ajouter">http://localhost/tutoriel-zf/public/index/ajouter</a>	dans IndexController::ajouterAction()
<a href="http://localhost/tutoriel-zf/public/index/modifier">http://localhost/tutoriel-zf/public/index/modifier</a>	dans IndexController::modifierAction()
<a href="http://localhost/tutoriel-zf/public/index/supprimer">http://localhost/tutoriel-zf/public/index/supprimer</a>	dans IndexController::supprimerAction()

 *Note du traducteur : J'ai traduit les noms des actions afin d'obtenir des URIs 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('/chemin/vers/fichiers-de-vue');
echo $view->render('view.php');
```

Il est évident que si nous devons 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}/{nom de l'action}.phtml` et le contenu (évalué/rendu) est au corps 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` :

tutoriel-zf/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 :

tutoriel-zf/application/views/scripts/index/index.phtml

```
<html>
<head>
<title><?php echo $this->escape($this->title); ?></title>
```

## tutoriel-zf/application/views/scripts/index/index.phtml

```
</head>
<body>
  <h1><?php echo $this->escape($this->title); ?></h1>
</body>
</html>
```

## tutoriel-zf/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>
```

## tutoriel-zf/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>
```

## tutoriel-zf/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. C'est un problème très classique, et le composant **Zend\_Layout** est conçu pour y remédier. Il nous permet de déplacer tout le code HTML des en-têtes et pieds de page vers un script de "mise en page" (*layout*) qui se charge alors d'inclure le code de la Vue en cours d'exécution.

Les modifications suivantes sont nécessaires. Premièrement, nous devons décider où conserver nos scripts de mise en page. L'emplacement recommandé est à l'intérieur du répertoire "application", il faut alors créer un répertoire "layouts" dans le répertoire "tutoriel-zf/application".

Nous devons initier le système Zend\_Layout dans le bootstrap index.php :

```
...
$frontController->throwExceptions(true);
$frontController->setControllerDirectory('./application/controllers');
Zend_Layout::startMvc(array('layoutPath'=> './application/layouts'));

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

La méthode **startMvc()** effectue quelques opérations en arrière-plan afin de mettre en place un plugin pour le front controller, afin de s'assurer que le composant `Zend_Layout` évalue le script de mise en page avec les scripts de Vue d'action à l'intérieur à la fin de la répartition.

Nous avons maintenant besoin d'un script de mise en page. Par défaut, il est appelé **layout.phtml** et il se situe dans le répertoire **"layouts"** :

```
tutoriel-zf/application/layouts/layout.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="fr" lang="fr">
<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">
    <h1><?php echo $this->escape($this->title); ?></h1>
    <?php echo $this->layout()->content; ?>
</div>
</body>
```

Remarquez que nous avons écrit notre code en XHTML et que c'est du code HTML standard pour afficher une page. Puisque le titre de la page dans la balise `<h1>` est utilisé dans toutes les pages, nous l'avons placé dans le fichier de mise en page et nous utilisons le *helper* **escape()** pour nous assurer qu'il est correctement encodé.

Pour obtenir que le script de l'action courante s'affiche, nous utilisons **echo** pour afficher le contenu de la variable **\$content** à l'aide du *helper* **layout()** : `echo $this->layout()->content;` Cela signifie que les scripts de Vue sont exécutés avant le script de mise en page.

Nous pouvons maintenant nous défaire des quatre scripts de vue puisque nous n'avons rien de particulier à y placer. Nous pouvons vider leur contenu.

Si vous essayez à nouveau les 4 URLs, vous ne devriez voir aucune différence ! La différence majeure est que, cette fois, tout le travail est effectué dans la mise en page.

## 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.

Les "assistants de Vue" (*view helpers*) sont situés dans le répertoire "application/views/helpers", sont appelés **{Nom de l'assistant}.php** (la première lettre doit être en majuscules) et la classe qui y réside doit être appelée "Zend\_View\_Helper\_{Nom de l'assistant}" (de nouveau, la première lettre est en majuscules). Il doit y avoir une fonction appelée "{nom de l'assistant}()" (première lettre en minuscules - n'oubliez pas !). Dans notre cas, le script s'appelle **BaseUrl.php** et il contient :

```
tutoriel-zf/application/views/helpers/BaseUrl.php
<?php
class Zend_View_Helper_BaseUrl
{
    function baseUrl()
    {
        $fc = Zend_Controller_Front::getInstance();
        return $fc->getBaseUrl();
    }
}
```

```
tutoriel-zf/application/views/helpers/BaseUrl.php
```

```
}
```

Ce n'est pas une méthode complexe. Nous récupérons simplement une instance du *front controller* et nous retournons la valeur de sa méthode **getBaseUrl()**.

Nous devons ajouter le fichier CSS à la section <head> du fichier **layout.phtml** :

```
tutoriel-zf/application/layouts/layout.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/css/site.css" />
</head>
...
```

Enfin, quelques styles :

```
tutoriel-zf/public/css/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 :


tutoriel-zf/application/config.ini

```
[general]
db.adapter = PDO_MYSQL
db.params.host = localhost
db.params.username = rob
db.params.password = 123456
db.params.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->params***.

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

tutoriel-zf/index.php

```
...
include "Zend/Loader.php";
Zend_Loader::registerAutoload();

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

// Mise en place du contrôleur
$frontController = Zend_Controller_Front::getInstance();
...
```

Nous chargeons la section "general" de "application/config.ini" dans l'objet **\$config**, puis nous assignons l'objet **\$config** au registre afin de pouvoir le réutiliser ailleurs dans l'application.

**i** 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) :

```
tutoriel-zf/index.php
...
$registry = Zend_Registry::getInstance();
$registry->set('config', $config);

// Mise en place de la BDD
$db = Zend_Db::factory($config->db);
Zend_Db_Table::setDefaultAdapter($db);

// Mise en place du contrôleur
$frontController = Zend_Controller_Front::getInstance();
...
```

Comme vous pouvez le voir, Zend\_Db a une méthode statique **factory()** qui interprète les données de l'objet **\$config->db** et qui instancie le bon adaptateur de bases de données à notre place.

## V-D - Créer la table

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

```
CREATE TABLE albums (
  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 albums (artist, title)
VALUES
  ('Duffy', 'Rockferry'),
  ('Van Morrison', 'Keep It Simple');
```

## 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 "Albums" 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 Albums dans le répertoire "application/models" :

tutoriel-zf/application/models/Albums.php

```
<?php

class Albums extends Zend_Db_Table
{
    protected $_name = 'albums';
}
```

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.

tutoriel-zf/application/controllers/IndexController.php

```
...
function indexAction()
{
    $this->view->title = "Mes albums";
    $album = new Albums();
    $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 :

tutoriel-zf/application/views/scripts/index/index.phtml

```
<p><a href="<?php echo $this->url(array('controller'=>'index',
    'action'=>'ajouter')); ?>">Ajouter un nouvel album</a></p>

<table>
<tr>
<th>Title</th>
<th>Artist</th>
<th>&nbsp;</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>
```

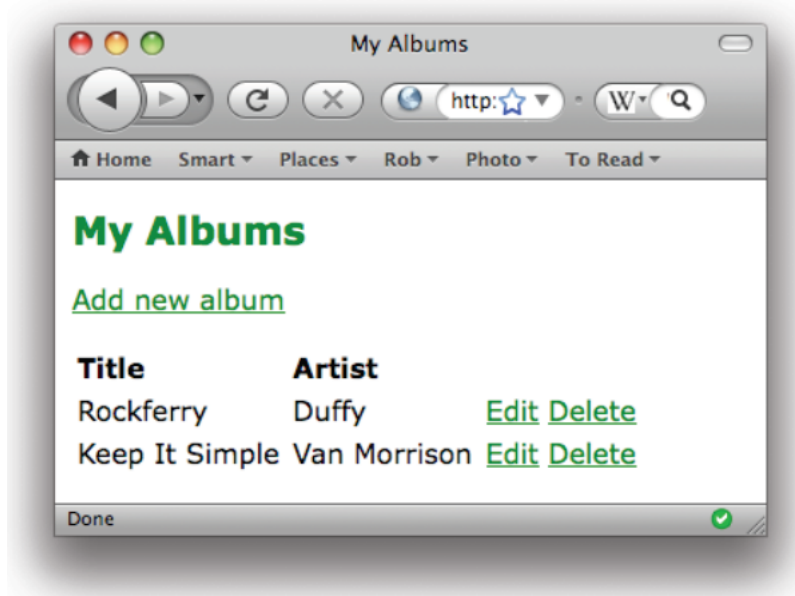
tutoriel-zf/application/views/scripts/index/index.phtml

```
<a href="<?php echo $this->url(array('controller'=>'index',
    'action'=>'modifier', 'id'=>$album->id));?>">Modifier</a>
<a href="<?php echo $this->url(array('controller'=>'index',
    'action'=>'supprimer', 'id'=>$album->id));?>">Supprimer</a>
</td>
</tr>
<?php endforeach; ?>
</table>
```

La première chose que nous faisons est de créer un lien pour ajouter un nouvel album. L'assistant `url()` est fourni avec le framework et crée une adresse avec la bonne URL de base. Nous lui donnons simplement les valeurs en paramètre et il trouve le reste.

Nous créons ensuite un tableau HTML pour afficher le titre et l'artiste, et pour proposer des liens permettant de modifier ou de supprimer l'enregistrement. Une boucle standard `foreach` est utilisée pour parcourir les albums, et nous utilisons la syntaxe alternative car elle est plus simple que d'essayer de faire correspondre des parenthèses. L'assistant `url()` est là aussi utilisé pour créer les liens.

<http://localhost/tutoriel-zf/> devrait maintenant afficher une jolie liste de (deux) albums, quelque chose comme :



## V-H - Ajouter des albums

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

### Il y a deux étapes à cette partie :

- Afficher un formulaire pour que l'utilisateur fournisse des détails ;
- Récupérer les informations et enregistrer en base.

Nous utiliserons **Zend\_Form** pour y parvenir. Ce composant nous permet de créer un formulaire et d'en valider le contenu. Nous créons une nouvelle classe **FormulaireAlbum** qui étend **Zend\_Form** pour définir notre formulaire :

tutoriel-zf/application/models/FormulaireAlbum.php

```
<?php

class FormulaireAlbum extends Zend_Form
{
```

## tutorial-zf/application/models/FormulaireAlbum.php

```

public function __construct($options = null)
{
    parent::__construct($options);
    $this->setName('album');

    $id = new Zend_Form_Element_Hidden('id');

    $artist = new Zend_Form_Element_Text('artist');
    $artist->setLabel('Artist');
    ->setRequired(true)
    ->addFilter('StripTags')
    ->addFilter('StringTrim')
    ->addValidator('NotEmpty');

    $title = new Zend_Form_Element_Text('title');
    $title->setLabel('Title');
    ->setRequired(true)
    ->addFilter('StripTags')
    ->addFilter('StringTrim')
    ->addValidator('NotEmpty');

    $submit = new Zend_Form_Element_Submit('submit');
    $submit->setAttrib('id', 'submitbutton');

    $this->addElements(array($id, $artist, $title, $submit));
}
    
```

Dans le constructeur, nous créons quatre éléments pour l'id, l'artiste, le titre et le bouton d'envoi. Chaque élément se voit affecté certains attributs, notamment le libellé à afficher. Pour les éléments textuels, nous ajoutons deux filtres StripTags et StringTrim pour supprimer le HTML et les espaces indésirables. Nous les mettons également comme "required" (*nécessaires*) et nous leur ajoutons un validateur NotEmpty (*non vide*) pour nous assurer que l'utilisateur entre véritablement l'information dont nous avons besoin.

Nous avons maintenant besoin d'afficher le formulaire et d'en récupérer les données. Cela se fait dans l'action **ajouterAction()** :

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

```

...
function ajouterAction()
{
    $this->view->title = "Ajouter un nouvel album";
    $form = new FormulaireAlbum();
    $form->submit->setLabel('Ajouter');
    $this->view->form = $form;
    if ($this->_request->isPost()) {
        $formData = $this->_request->getPost();
        if ($form->isValid($formData)) {
            $albums = new Albums();
            $row = $albums->createRow();
            $row->artist = $form->getValue('artist');
            $row->title = $form->getValue('title');
            $row->save();

            $this->_redirect('/');
        } else {
            $form->populate($formData);
        }
    }
}
...
    
```

Examinons ceci en plus de détails :

```
$form = new FormulaireAlbum();
$form->submit->setLabel('Ajouter');
$this->view->form = $form;
```

Nous créons notre FormulaireAlbum, nous donnons le libellé "Ajouter" au bouton d'envoi et ensuite nous assignons le formulaire à la Vue pour affichage.

```
if ($this->_request->isPost()) {
    $formData = $this->_request->getPost();
    if ($form->isValid($formData)) {
```

Si la méthode isPost() de l'objet Request renvoie TRUE, alors le formulaire a été envoyé. Nous récupérons les données par la méthode getPost() et nous les vérifions avec isValid().

```
$albums = new Albums();
$row = $albums->createRow();
$row->artist = $form->getValue('artist');
$row->title = $form->getValue('title');
$row->save();

$this->_redirect('/');
```

Si le formulaire est valide, alors nousinstancions la classe modèle Albums et nous utilisons **createRow()** pour récupérer un nouvel enregistrement et en remplir le titre et l'artiste avant d'enregistrer. Après avoir sauvegardé le nouvel enregistrement en base, nous revenons à la page d'accueil avec la méthode **\_redirect()** du contrôleur.

```
    } else {
        $form->populate($formData);
    }
}
```

Si les données du formulaire ne sont pas valides, alors nous le remplissons avec les données fournies et nous affichons à nouveau.

Nous avons maintenant besoin d'évaluer le formulaire dans le script ajouter.phtml :

```
tutoriel-zf/application/views/scripts/index/ajouter.phtml
<?php echo $this->form ;?>
```

Comme vous pouvez le voir, afficher un formulaire est très simple puisqu'il sait comment se prendre en charge.

## V-I - Modifier un album

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

```
tutoriel-zf/application/controllers/IndexController.php
...
function modifierAction()
{
    $this->view->title = "Modifier un album";
    $form = new FormulaireAlbum();
```

tutoriel-zf/application/controllers/IndexController.php

```

$form->submit->setLabel('Enregistrer');
$this->view->form = $form;

if ($this->_request->isPost()) {
    $formData = $this->_request->getPost();
    if ($form->isValid($formData)) {
        $albums = new Albums();
        $id = (int)$form->getValue('id');
        $row = $albums->fetchRow('id='.$id);
        $row->artist = $form->getValue('artist');
        $row->title = $form->getValue('title');
        $row->save();

        $this->_redirect('/');
    } else {
        $form->populate($formData);
    }
} else {
    // L'id de l'album est attendu dans $params['id']
    $id = (int)$this->_request->getParam('id', 0);
    if ($id > 0) {
        $albums = new Albums();
        $album = $albums->fetchRow('id='.$id);
        $form->populate($album->toArray());
    }
}
...

```

Voyons les différences avec un ajout. Premièrement, lorsque nous affichons le formulaire, nous avons besoin des données en base et de remplir le formulaire avec :

```

$id = (int)$this->_request->getParam('id', 0);
if ($id > 0) {
    $albums = new Albums();
    $album = $albums->fetchRow('id='.$id);
    $form->populate($album->toArray());
}

```

Ce n'est exécuté que si la requête est bien POST, et utilise le modèle pour récupérer l'enregistrement depuis la BDD. La classe **Zend\_Db\_Table\_Row** a une méthode **toArray()** que nous pouvons utiliser directement pour remplir le formulaire.

Finalement, nous avons besoin de sauvegarder les données dans le bon enregistrement. On y parvient en récupérant l'enregistrement puis en le sauvegardant de nouveau :

```

$albums = new Albums();
$id = (int)$form->getValue('id');
$row = $albums->fetchRow('id='.$id);
$row->artist = $form->getValue('artist');
$row->title = $form->getValue('title');
$row->save();

```

Le code de modifier.phtml est le même que pour ajouter.phtml :

tutoriel-zf/application/views/scripts/index/modifier.phtml

```
<?php echo $this->form ;?>
```

Vous devriez désormais être en mesure d'ajouter et de modifier des enregistrements.

## 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.

Nous allons afficher un formulaire de confirmation à l'utilisateur et, s'il clique "oui", nous effectuerons la suppression. Puisque ce formulaire est trivial, nous allons simplement le faire en HTML dans le script de Vue.

Commençons par le code de l'action :

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

    if ($this->_request->isPost()) {
        $id = (int)$this->_request->getPost('id');
        $del = $this->_request->getPost('del');
        if ($del == 'Oui' && $id > 0) {
            $albums = new Albums();
            $where = 'id = ' . $id;
            $albums->delete($where);
        }
        $this->_redirect('/');
    } else {
        $id = (int)$this->_request->getParam('id');
        if ($id > 0) {
            $albums = new Albums();
            $this->view->album = $albums->fetchRow('id='.$id);
        }
    }
}
...
```

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 Albums. Tout comme l'ajout et la modification, la suppression est faite au moyen d'un appel à **Zend\_Db\_Table::delete()**. Si la requête n'est pas POST, alors nous recherchons un paramètre "id", nous récupérons le bon enregistrement depuis la BDD et nous assignons la Vue.

Le gabarit est un simple formulaire :

```
tutoriel-zf/application/views/scripts/index/supprimer.phtml
<?php if ($this->album) :?>
<p>Êtes-vous sûr de vouloir supprimer
    '<?php echo $this->escape($this->album->title); ?>' par
    '<?php echo $this->escape($this->album->artist); ?>' ?
</p>
<form action="<?php echo $this->url(array('action'=>'supprimer')); ?>" method="post">
<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>Impossible de trouver l'album.</p>
<?php endif;?>
```

Nous affichons dans le script un message de confirmation ainsi qu'un formulaire avec les boutons "Oui" et "Non". Dans l'action, nous avons comparé avec la valeur "Oui" au moment de la suppression.

C'est tout. Vous avez maintenant une application pleinement fonctionnelle.

## 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 :




tutoriel-zf/index.php

```
...
// setup controller
$frontController = Zend_Controller_Front::getInstance();
$frontController->throwExceptions(true);
$frontController->setControllerDirectory('./application/controllers');
$frontController->setBaseUrl('/mysubdir/tutoriel-zf/public');
Zend_Layout::startMvc(array('layoutPath'=> './application/layouts'));
...
```

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

### 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](mailto: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](#).